Integrated analysis of software product lines

A constraint based framework for consistency, liveness, and commonness checking

Jean-Vivien Millo, Swarup K Mohalik, S. Ramesh
India Science Lab, General Motors Global Research and Development
GM Technical Center India Pvt Ltd, Creator Building
International Tech Park Ltd., Whitefield Road, Bangalore - 560 066, India.
{jean-vivien.millo, swarup.mohalik, ramesh.s}@gm.com

ABSTRACT

Software Product Line (SPL) is a software development framework to jointly design a family of closely related software products in an efficient and cost-effective manner. In order to separate the concerns and handle complexity, designers usually project the SPL along different perspectives such as feature, architecture and behaviour. Each perspective deals with variability of a set of artifacts and variability constraints among them. SPL designers attempt to ensure the *consistency* of the individual perspectives and the SPL as a whole. They are also interested in finding the elements *common* to all products and the *live* elements (used in at least one product).

In the literature, most of the works focus on a single perspective and address the above-mentioned problems within single perspectives. There have also been attempts to express the variability of different perspectives within the feature perspective. However, since the different perspectives have different intents, coercing them into a single perspective may result in unnatural constructs in the feature perspective. Hence, it is better to keep the perspectives separate. However, in any SPL, the perspectives are closely related through an implementability relation or through constraints arising from design or business reasons. We call this the traceability aspect, which mandates an integrated analysis of the different perspectives.

In this paper, we propose a constraint-based framework where variability and traceability constraints can be uniformly expressed, at the same time keeping the different intents of perspectives intact. We describe how the consistency, liveness, and commonness problems can be reduced to problems of constraint solving. Through a realistic case study, we provide some evidence that the constraint-based framework is expressive and scalable to large SPLs.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods*

General Terms

Verification

ISEC'11, February 23-27, 2011 Thiruvananthapuram, Kerala, India

Keywords

Consistency, liveness, commonness, variability, traceability

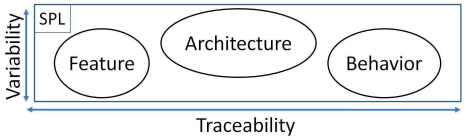
1. INTRODUCTION

Software Product Line (SPL) is a software development framework to jointly design a family of closely related software *products* (also called *variants*) in an efficient and cost-effective manner. All the variants in the family are composed from a single set of components developed once for the entire family.

Since the specification and implementation of an SPL for industrial products is complex, in order to separate the concerns and handle complexity, designers usually project the SPL along different perspectives. A typical methodology [20] prescribes three perspectives: feature, architecture and behaviour. The feature perspective lists the features and functionalities of the products of the SPL. Features are implemented by components in the behaviour perspective. The architecture perspective specifies how the components are to be organized and interconnected to achieve the feature requirements.

SPLs require well-defined mathematical models to support a variety of analysis. A number of such models have been proposed in the literature to express the different perspectives of an SPL (e.g. features, architectures and behaviors) [5, 4, 19, 20]. Each perspective deals with a specific set of elements and their relations. A major source of complexity of these models is the *variability* [22] of the product-line elements which captures (1) the fact that the elements may be mandatory, alternative or optional and (2) the correlation between elements. Correlation relations constrain the presence (absence) of an element through the presence (absence) of one or more other elements. In addition, the elements of different perspectives are connected together through what are called *traceability* constraints. The variability and traceability constraints together restrict the space of products derivable from an SPL. A schematic of the perspectives and the constraints is given in Figure 1. As depicted, the variability constraints relate elements *within* a perspective, and the traceability constraints relate elements *across* perspectives.

Figure 1: Schematic of perspectives and constraints of an SPL



One of the most important problems facing an SPL designer is the *consistency* problem, namely, to find if the SPL is consistent i.e. if

there are possible products satisfying the variability and traceability constraints. Further, if the SPL is inconsistent then one wants to find the source of inconsistency to debug the SPL. The designers also may want to find that the elements of the SPL are not superfluous. An element is superfluous (or *dead*) if it is not used in any product derivable from the SPL. On the other hand, an element is called *common* if it is necessary for all the products of the SPL. Finding the dead and common elements of an SPL are called, respectively, the *liveness* and the *commonness* problem.

In the literature, the problems of consistency, liveness, and commonness have mostly been addressed within individual perspectives. Except in [25, 18], there is no work treating the variability and traceability across all the perspectives of an SPL in an integrated fashion. From the point of view of a complete product derivation, such integrated analysis are necessary. Further, one should have automated solutions for the consistency, liveness, and commonness problems since real-life SPLs involve thousands of elements and manual methods are not scalable. Another notable point is that existing standard models for the feature perspective do not have a clean way of expressing complex constraints among feature elements. We address all these issues, namely, expressibility, integrated solution and automated methods in this paper.

1.1 Related Work

Our work is closely related to many research areas of Software Product Line Engineering. In the following, we give an overview of the state-of-the-art.

Feature modeling. Modeling the feature perspective of SPLs has garnered a lot of attention in the research community. Krzysztof Czarnecki [5] introduced a feature model which is now considered standard. Don Batory [2] has developed a method based on stepwise refinement to automatically generate product code from a selection of features from a feature model which is clearly inspired from Czarnecki. In addition to these models, David Benavides et al. [35] perform (in)consistency checking of feature models.

Architecture modeling. Many architecture models [10, 31, 13, 28] exist but Tommi Myllymaki [30] points out the lack of architectural models that can explicitly handle variability. Only EASEL (MENAGE) by André van der Hoek [13] is able to do it. Similarly, Eiji Adachi et al. [1] combine the ACME architecture model [10] with an independent variability model.

Behavior modeling. Ulrik Nyman [19] has defined the I/O Modal Automata (IOMA) to express in one automaton all the possible behaviors of a component. Similarly, Alessandro Fantechi and Stefania Gnesi [8] have defined the GEMTS model which has more general variability constraints than IOMA using the *at least* and *at most* operators. Kathrin Scheidemann et al. [12] present a behavioral model (PL-CCS) based upon CCS.

Variability modeling. Many works [11, 6, 9, 32, 21] discuss about variability problems and present sound variability models. Klaus Pohl et al. defined the OVM (Orthogonal Variability Model) [18] which extracts variability from feature model only. Fabricia Roos-Frantz compares the OVM with Czarnecki feature model in [24]. Finally, Deepak Dhungana [7] introduces a decision-oriented variability model that links together the variability of the feature perspective with the variability of the architecture perspective.

Traceability modeling. Many traceability models [23, 25, 26] link the features to the components of the architecture. Some other approaches [36, 27] have linked requirements to implementations through feature and architecture.

Full SPL modeling. A full SPL model is a model or a set of models that includes all the perspectives of an SPL from requirement to code. This domain is dominated by two tools: GEARS [16] by Charles W. Krueger and *Pure::Variant* [3] by Danilo Beuche. Our approach is different from these because our goal is formal verification of consistency, liveness, and commonness and not synthesis of products as in these works, though, as a side-effect of consistency checking, one can get products.

Verification in SPL. Kathi Fisler et al. [15] have developed a verification method based on three-valued model checking of automata defined using step-wise refinement. Each step of the refinement adds the behavior of another feature to the automaton. Don Batory [2] presents a method to check the consistency of feature model. Robin Laney et al. [17] introduce a behavior consistency checker for composition of features. Mikolás Janota et al. [14] present a survey of the usage of formal methods in the field of software product line. The authors conclude that formal methods are getting more and more used but the consistency checking problem of the SPL remains a challenge. In view of the constraint based frameworks, [2] deals with only the feature perspective. In [25], the authors handle feature and architecture (but not the behaviour) perspective in a theorem proving framework for first-order logic. In contrast, we analyze feature, architecture and behavior perspectives as well as traceability constraints in a simpler framework. It is possible that some traceability constraints as in [25] will lead to large number of propositional constraints in our framework. However, the simple structure of the constraints allow them to be solved by state-of-the-art constraint solvers with small amount of resources (time and memory).

1.2 Our Contribution

In this paper, we solve the problems of consistency, liveness, and commonness checking for an SPL considering all the perspectives jointly and modeling the variability and traceability information in a constraint based framework. Different models used for different perspectives of an SPL can be represented as constraints, and constraint solving can be used to find the solutions to the consistency, liveness, and commonness problems.

We observe that the variability and traceability constraints play a crucial role in limiting the space of SPL variants. Another advantage of the joint analysis is that one gets valid product configurations as solution to the consistency problem.

We also present a realistic SPL of an entry control system for cars using a feature model, an architecture model called Model Architecture Model (MaM) close to EASEL and a behavior model called GEMTS. We show how the variability constraints can be extracted out of these models automatically. We then illustrate the impact of

variability and traceability constraints on the consistency, liveness, and commonness problems.

1.3 Organization of the paper

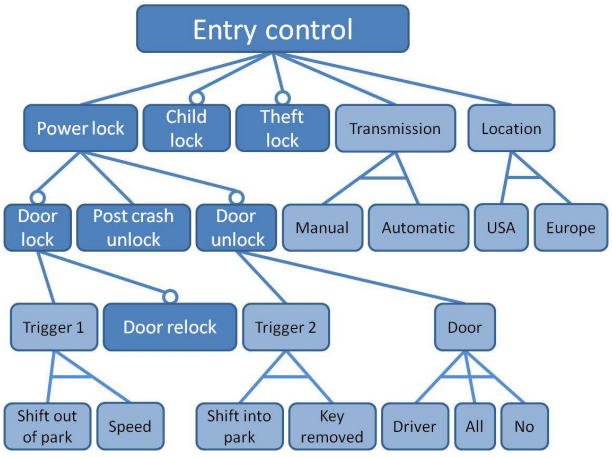
The rest of the paper is organized as follows: we describe the case study of an entry control system in Section 2. The motivation of our approach is given in Section 3. Section 4 specifies the constraints-based framework for SPL. We formalize the problems of consistency, liveness, and commonness in Section 5 and propose the solutions for the problems through constraint solving. Section 6 presents our experimental results on the case study of Section 2. Section 7 concludes this paper with an outline of open issues.

2. THE ENTRY CONTROL PRODUCT LINE

We introduce here a case study of Entry Control Product Line (ECPL) used in the automotive industry. It will be used all along the paper to motivate and illustrate our work. The entry control system comprises all the features involved in the management of locks of cars. In this study, we focus on the following features:

- Power lock: this is the basic locking functionality which manages the locking/unlocking according to key button press and courtesy switch press.
- Door lock: automates the locking of doors.
- Door unlock: automates the unlocking of door(s).
- Door relock: performs automatic relocking of doors in case of pick up/drop and drive.
- Post crash unlock: causes the unlocking of doors in a post crash situation.
- Child lock: secures rear doors for children.
- Theft lock: secures the doors from external opening with an additional lock.

Figure 2: The feature model of the ECPL



The feature perspective of the ECPL. Figure 2 presents the feature model of the ECPL (a la Czarnecki). The dark gray boxes are features of the ECPL. The light gray boxes are parameters modeled as features. Child lock and Theft lock are optional features. The Door lock feature is optional and can be triggered either when the gear is shifted out of park or when the car speed reaches a predefined value. The Door relock feature is optional. Finally, the door unlock feature is also optional and can be triggered either when the gear is shifted into park or when the car key is removed from ignition. The unlock action opens the driver door only, all the doors, or no door depending upon a calibration. The car should have either a manual or an automatic transmission and is destined for either American or European market.

Inputs/Outputs of features. Table 1 presents the relations between features and Inputs/Outputs of the ECPL.

Table 1: The relations between features and Inputs/Outputs of the ECPL

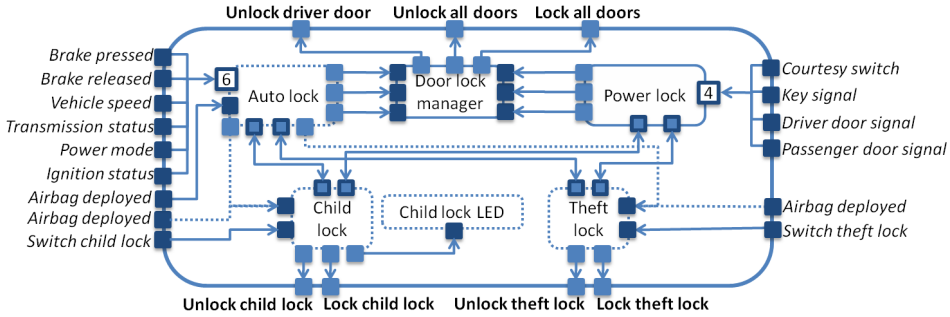
Features	Inputs	Outputs
Power lock	*Courtesy switch* *Key signal* *Driver door signal* *Passenger door signal*	**Unlock driver door** **Unlock all doors** **Lock all doors**
Child lock	*Airbag deployed* *Switch child lock*	**Unlock child lock** **Lock child lock**
Theft lock	*Airbag deployed* *Switch theft lock*	**Unlock theft lock** **Lock theft lock**
Post crash unlock	*Airbag deployed*	**Unlock all doors**
Door unlock	*Transmission status* *Power mode* *Ignition status*	**Unlock driver door** **Unlock all doors**
Door lock	*Vehicle speed* *Transmission status* *Power mode*	**Lock all doors**
Door relock	*Brake pressed* *Brake released* *Vehicle speed* *Power mode*	**Lock all doors**

The architecture perspective of the ECPL. Figure 3 represents an architecture model presented in a notation called Modal Architectural Model (Appendix A). It is a simplified form of EASEL [13] and yet preserves the essential notion of variability central to the product line. The model is composed of six components: $Power\ lock$, $Door\ lock\ manager$, $Auto\ lock$, $Child\ lock$, $Theft\ lock$, and $Child\ lock\ LED$. The first two items are mandatory but the other four items are optional (denoted in the diagram by dotted boxes). The full system has 13 "in" ports (dark gray squares) and 7 "out" ports (light gray squares). For clarity, we have grouped 6 "in" ports of the feature $Auto\ lock$ and 4 "in" ports of the feature $Power\ lock$ in two graphical objects (squares with dark border). The port *Airbag deployed* is replicated again for clarity of the figure.

The $Auto\ lock$ component requires six global input signals plus the airbag signal, it exchanges status signal with $Child\ lock$ and $Theft\ lock$ components, provides lock/unlock action signals to the $Door\ lock\ manager$ and may provide the airbag signal to $Child\ lock$ and $Theft\ lock$ components. The $Power\ lock$ component requires the four global input signals, it exchanges status signal with $Child\ lock$ and $Theft\ lock$ components, and provides lock/unlock action signals to the $Door\ lock\ manager$.

The $Door\ lock\ manager$ component arbitrates between the lock/unlock action signals from $Auto\ lock$ and $Power\ lock$ and forwards them to the global outputs. $Child\ lock$ and $Theft\ lock$ components require their respective Switch signal, exchange status signal with $Auto\ lock$ and $Power\ lock$ and provide respective lock/unlock actions. These components may receive the airbag signal directly from the global inputs or through the $Auto\ lock$

Figure 3: The Modal Architecture Model of the ECPL

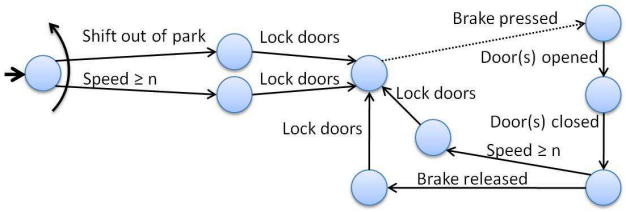


component. This variability is the artifact of the architecture and its choice depends upon a design decision. The $Child\ lock\ LED$ drives the status of the LED associated to the $Child\ lock$ feature.

The behavior perspective of the ECPL. Figures 4, 5 and 6 describe the behaviours of (1) the door lock feature and the door relock feature, (2) the door unlock feature and (3) the post crash unlock feature using automata. The behaviors of other features are not presented here because of their lack of variability and also because of space limitation.

The bent arrows crossing transitions introduce variability in the automata. Each automaton represents many possible behaviors that a feature can take. A product of the ECPL contains exactly one transition from those crossed by the same bent arrow. The transition is dotted when optional.

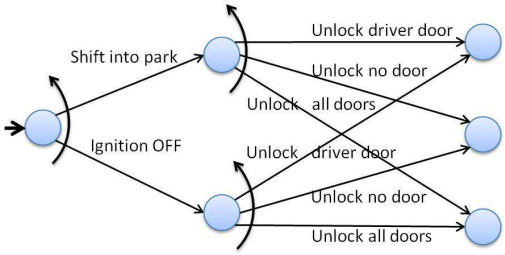
Figure 4: The automaton of Door lock and Door relock features in the Auto lock component



The automaton of the door lock feature in Figure 4 starts from an initial state where the Door lock feature is enabled, the engine is running and all doors are closed. Depending upon the transmission configuration (automatic or manual), and user preferences (see Section 6.4 for details), all doors lock when either the gear is shifted out of park *or* the car speed exceeds a calibrated value (n). At this state, Door relock is triggered if the driver brakes, then at least one door is opened and closed. All doors will lock again when either the brake is released or car speed exceeds n.

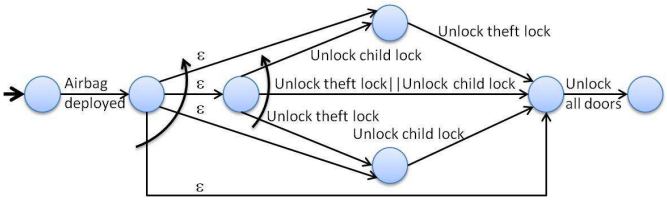
The Door unlock feature's automaton in Figure 5 starts from an initial state where the Door unlock feature is enabled, the engine is running and all doors are closed. According to the transmission configuration or to the user preference (see Section 6.4 for details), the doors are unlocked when either the gear is shifted into park *or* the car key is removed from ignition. The doors unlocked by the unlock action can be the driver door only, all the doors, or no door.

Figure 5: The automaton of Door unlock feature in the Auto lock component



The Post crash unlock feature's automaton in Figure 6 starts from an initial state where a crash has occurred. First, the airbags are deployed. Then, according to the presence of Child Lock and/or Theft Lock, these two locks are opened and then, all doors are unlocked. If both Child Lock and Theft Lock are present in the product, there is no specification about the order in which they are supposed to be unlocked. Any order can be followed or the two unlock actions can be done in parallel. This choice of order of execution is an example of variability specific to the behavior perspective.

Figure 6: The automaton of Post crash unlock feature in the Auto lock component



The traceability constraints of the ECPL. Table 2 presents the traceability constraints between the features and the components of the ECPL.

Table 2: Traceability constraints of the ECPL

Feature	Component
Power lock	Power lock& Door lock manager
Child lock	Child lock & Child lock LED
Theft lock	Theft lock
Post crash Unlock	Auto lock
Door lock	Auto lock
Door unlock	Auto lock
Door relock	Auto lock

3. PERSPECTIVES AND INTEGRATED ANALYSIS

As we have illustrated in Section 2, an SPL can be expressed using different perspectives such as feature, architecture, and behavior perspectives. In the literature, there are different modeling paradigms for different perspectives: Czarnecki feature model focuses on the feature perspective [5], EASEL [13] focus on the architecture perspective, and GEMTS [8] focuses on the behavior perspective.

Most of these models are informal as they have only graphical representation to ease the understanding of the SPL's perspectives. This attribute limits their capacity of representation of constraints. Consequently some constraints belonging to the model's perspective cannot be expressed. For example, in the ECPL, the following constraint is not part of the Figure 2: *if a product is made for the* USA, *the door lock feature must be present.* In [8], the authors introduce an additional language called ACTL to represent constraints that cannot be expressed directly within a GEMTS instance.

In every SPL, there are some constraints on the elements across different perspectives. In ECPL, the presence of the Theft lock feature constraints the presence of the Theft lock component and the behavior of the Post crash unlock feature. We call them traceability constraints (from feature to components, from feature to behavior, etc.). Of course, a product extracted from a SPL is valid only if it respects the variability constraints from every perspectives of the SPL as well as the traceability constraints.

Representation of different perspectives using different formalisms creates difficulties in carrying out an integrated analysis of the entire SPL. In order to check the consistency of the SPL and to find the live and common elements we need a uniform framework.

In the following Section, we propose a constraint based framework to express uniformly all the constraints in an integrated fashion regardless of the perspective(s) they come from. Then, the problems of consistency, liveness and commonness of the elements of the SPL are reduced to Constraint Satisfaction Problem (CSP) and can be solved using off-the-shelf tools such as YICES [34] or BDDSOLVE [33].

4. A CONSTRAINT BASED FRAMEWORK

In this section, we give formal definitions of the three perspectives, namely, feature, architecture and behaviour, through the set of elements in each perspective and the variability constraints. The traceability constraints are defined over the elements of every perspectives. An SPL is then defined through these perspectives plus the traceability constraints.

The most important aspect of a product line is the variability, which is reflected in each of the perspectives. Variability is introduced by the choice between elements that are mandatory, optional or alternatives. An element or a group of elements for which these choices are available is called a *variation point*. In the feature model of the ECPL, the transmission is a variation point because it can be either automatic or manual. Door relock is optional, this is also a variation point. There are additional constraints correlating different elements within each perspective coming from business reasons (bundling certain features) or from design constraints. These choices are captured through the *variability constraints*.

The following constraint language is used to model commonly known constraints in SPL engineering. Let V be a finite set of boolean valued elements. We denote by C_V a set of boolean expressions over V. A ground assignment is a function $\rho : V \rightarrow Bool$. It can be equivalently expressed as a subset of $V : V' = \{v \in V | \rho(v) = true\}$. A ground assignment V' satisfies the set of constraints C_V (denoted $V' \models C_V$) if all the constraints in C_V evaluate to true with the ground assignment V'. C_V is *consistent* if there is a satisfying ground assignment and it is called *inconsistent* otherwise.

Each element e in the perspectives of the SPL is modeled as a boolean valued proposition e_p. When it is clear in the context, we treat each element e as the corresponding propositional variable e_p so as to simplify the notation. Each perspective is given as a set of constraints C on its elements. A variation point is an element which is not assigned the value $true$ or $false$. A satisfying ground assignment ρ for the constraints C fixes the values of the elements and thus defines a possible product of the perspective. In this product, an element e is present iff $\rho(e_p) = true$.

4.1 Feature perspective

We define a feature of a system as a set of requirements on the system behaviour identified by a keyword (name of the feature). The first level of variability in a feature model comes from the possibility of absence/presence of a feature. In this paper, we also model the inputs and outputs of the features as elements of the feature perspective. This gives us more variability than the feature names alone and has more meaningful traceability with the architecture perspective. In ECPL, Table 1 gives the inputs and outputs required by each feature. This information is extracted from the requirements.

A feature f is defined as a triple $\langle f, Input_f, Output_f \rangle$ where f is the name of the feature, $Input_f$ and $Output_f$ are respectively finite sets of input and output variable names.

DEFINITION 1 (FEATURE PERSPECTIVE). *The feature perspective F is a tuple $\langle F, C_F \rangle$ where F is a set of features and C_F is a set of constraints. Let E_F be the set of elements $\bigcup_{f \in F}(\{f\} \cup Input_f \cup Output_f)$. C_F is defined over E_F and includes the following:*

- *These constraints capture the fact that if a feature is present, then so are all its inputs and outputs.* $f \Rightarrow \bigwedge_{i \in Input_f} i$, $f \Rightarrow \bigwedge_{o \in Output_f} o$,
- *In contrast, if an input (output) is present, then so is at least a feature that requires it.* $i \Rightarrow \bigvee_{f \in F / i \in Input_f} f$, $o \Rightarrow \bigvee_{f \in F / o \in Output_f} f$.

In the feature perspective of ECPL, the set of features contains all the elements of the Figure 2 plus the inputs and outputs of the Table 1. The set of constraints includes items such as $Door\ relock \Rightarrow Door\ lock$ and $Post\ crash\ unlock \Rightarrow Airbag\ deployed \wedge Unlock\ all\ doors$.

4.2 Architecture perspective

In an SPL, every product has the same architectural skeleton but some variations occur from product to product. In the architecture perspective of the ECPL, the skeleton is formed by the components $Door\ lock\ manager$ and $Power\ lock$ which are mandatory for every product (Figure 3). The interconnections among them and with the environment are also part of the skeleton. All the other elements form the variable part of the ECPL.

Let C be the set of components. A component $c \in C$ is a tuple $\langle name, interface \rangle$ where:

- $name$ is the name of the component.
- $interface$ is a set of $ports$. $port \in interface$ is a tuple $\langle p\_name, direction \rangle$ where p_name is the name of the port and $direction \in \{in, out, in\_out\}$.

We designate a special component called *root* for the purpose of defining the interface of the architecture with the environment.

Let I be the set of possible interconnections between components of C. An interconnection $i \in I$ is defined as a couple (p_1, p_2) where:

- p_1 is a port such that, $p_1.direction \in \{out, in\_out\}$, $p_1 \in c_1.interface$ and $c_1 \in C$.
- p_2 is a port such that, $p_2.direction \in \{in, in\_out\}$, $p_2 \in c_2.interface$ and $c_2 \in C$.
- $p_1 \neq p_2$.

DEFINITION 2 (ARCHITECTURE PERSPECTIVE). *An architecture perspective is a tuple $\langle (C, I), C_A \rangle$ where C is a set of components including the root component, and I is a set of interconnections. Elements of the architecture perspective are defined as $E_A = C \cup I$. C_A is a set of constraints defined over E_A.*

In the ECPL, the set of components contains the 6 components of the Figure 3 plus the root component. The set of interconnections contains the 35 interconnections of the same figure. One of the constraints in E_A relates the presence of the Child lock LED component to the presence of the Child lock component: $Child\ lock\ LED\ cmp \Leftrightarrow Child\ lock\ cmp$.

4.3 Behavior Perspective

The behaviour perspective of an SPL represents the behaviour of features using high level representation such as an automaton. In the ECPL, The figures 4, 5, and 6 present respectively the behavior of the features $Doorlock$ and $Door\ relock$, $Door\ unlock$, and $Post\ crash\ unlock$. The variability in this perspective captures the different possible behaviors of a given feature. Some transitions of the automaton model of a feature can be mandatory, optional, or alternative. For example, in Figure 4, the two outgoing transitions from the initial state are alternatives: only one of them is implemented in a product. The variability constraints relate either the presence of transitions from the same automaton or the presence of transitions from different automata. In the ECPL, one can express constraints such as "if the trigger of $Door\ lock$ is $Shift\ out\ of\ park$, the trigger of $Door\ unlock$ is $Shift\ into\ park$" (This specific constraint is not part of our case study even though we are able to model it).

DEFINITION 3 (BEHAVIOUR PERSPECTIVE). *The behaviour perspective of an SPL is defined as a tuple $\langle B, C_B \rangle$, where B is a set of automata. Each automaton is associated with one or many features. C_B is a set of constraints. The elements of the behaviour perspective are all the transitions from the automata of the SPL i.e. $E_B = \bigcup_{b \in B} T_b$ (where T_b is the set of transitions of the automata b). C_B is defined over E_B.*

In the behavior perspective of the ECPL, the set of elements is formed by all the transitions of figures 4, 5, and 6. For example, $Door\ lock\ trigger \Rightarrow Shift\ out\ of\ park \oplus Speed \geq n$ is one of the constraints in C_B (here, $\oplus$ is the "exclusive or" operation).

4.4 Traceability Constraints

Each feature may require one or more components for its implementation. On the other hand, each component can also be used to implement one or many features. In addition, a feature may be implemented by one or many automata from the behaviour perspective. Thus, there are relations across elements from different perspectives necessary for implementing products of an SPL. The set of traceability constraints C_T capture this "implements" relation across perspectives. There may be variability in the traceability relations themselves e.g. features can be satisfied by many alternative choices of components.

DEFINITION 4 (TRACEABILITY CONSTRAINTS). *The set of traceability constraints is a set of constraints over $E_F \cup E_A \cup E_B$, each constraint necessarily involving elements from more than one perspective (otherwise they would be part of the variability constraints)*

In the ECPL, Table 2 gives the traceability constraints from features to components. The first line of the table corresponds to the following constraint: $Power\ lock \Rightarrow Power\ lock\ cmp\ \&\ Door\ lock\ manager\ cmp$.

4.5 SPL

An SPL integrates the feature, architecture and behaviour perspectives, each comprising its variability constraints. In addition, the traceability constraints provide the link among the elements across different perspectives. Formally,

DEFINITION 5 (SPL). *An SPL is defined as a tuple $\langle FP, AP, BP, C_T \rangle$ where $FP = \langle F, C_F \rangle$ is a feature perspective, $AP = \langle A, C_A \rangle$ is an architecture perspective and $BP = \langle B, C_B \rangle$ is a behaviour perspective and C_T is a set of traceability constraints.*

4.6 Product

A feature product is a ground assignment of F satisfying the constraints C_F. Definition of architecture and behaviour products are

similar. A product of SPL consists of a feature product, an architecture product, and a behaviour product which also meet the traceability constraints C_T. Formally,

DEFINITION 6 (PRODUCT). *A product of an SPL is a tuple* $\langle F', A', B' \rangle$ *such that*

- $F' \models C_F$
- $A' \models C_A$
- $B' \models C_B$, *and*
- $F' \cup A' \cup B' \models C_T$

The set of all consistent products of an SPL is denoted as $PROD(SPL)$.

In ECPL, the following set of features: $\{ Power\,lock, Post\,Crash\,unlock, Transmission, Manual, Location, USA \}$ satisfies the set of constraints C_F. This is the smallest feature product in ECPL. If we associate to this feature product an architecture product and a behavior product that, all together, satisfy the traceability constraints, we get a product.

5. CONSISTENCY, LIVENESS, AND COMMONNESS

Let $C_{SPL} = C_F \cup C_A \cup C_B \cup C_T$, be the set comprising all the variability constraints from the perspectives and the traceability constraints. Also, let $E_{SPL} = E_F \cup E_A \cup E_B$ be the set of all elements of the SPL(features, components, interconnections and transitions).

Given the constraint-based model of an SPL as above, one can define the consistency, liveness and commonness as follows.

1. An SPL is said to be consistent iff $PROD(SPL)$ is non-empty i.e. there exists at least one product in the family.
2. An element $e \in E_{SPL}$ is live if $\exists P \in PROD(SPL)$ such that $e \in P$ i.e. it is contained in a product of SPL. It is called *dead* or superfluous otherwise.
3. An element $e \in E_{SPL}$ is common if $\forall P \in PROD(SPL)$ such that $e \in P$ i.e. it is contained in all the products of SPL.

The consistency, liveness, and commonness problems of elements of E_{SPL} are defined as:

- **Consistency problem:** Given an SPL, find if it is consistent.
- **Liveness problem:** Given an SPL, find the set of alive and dead elements.
- **Commonness problem:** Given an SPL, find the set of common elements.

The basic solutions for the problems use the results of the following proposition which is a simple consequence of the definition above.

PROPOSITION 7. *The following statements hold:*

- $PROD(SPL)$ *is non-empty iff* C_{SPL} *is satisfiable.*
- $e \in E_{SPL}$ *is live iff* $C_{SPL} \cup \{e\}$ *is satisfiable.*
- $e \in E_{SPL}$ *is common iff* $C_{SPL} \cup \{\neg e\}$ *is unsatisfiable.*

Thus, the consistency problem can be solved by checking the satisfiability of the set of constraints C_{SPL} using a constraint solver. One can find the set of live elements by iterating through the elements and checking their liveness (resp. commonness) through Proposition 7.2 (resp. Proposition 7.3). Computationally better algorithms for checking liveness and commonness are presented respectively in Algorithms 1 and 2.

Algorithm 1 checks whether elements of E_{SPL} are live or not.

Input: SPL
Output: E_{Live}, the set of live elements of SPL and E_{Dead}, the set of dead elements
E_{live} =Ø
E_{dead} =Ø
repeat
  Select $e \in E_{SPL} \setminus (E_{live} \cup E_{dead})$
  if $C_{SPL} \cup \{e\}$ is not satisfiable **then**
    {We add e to the set of dead elements.}
    $E_{dead} = E_{dead} \cup \{e\}$
  else
    Generate E_g, a satisfying ground assignment for $C_{SPL} \cup \{e\}$.
    {Obviously, the elements in E_g are live and instead of a single element, we possibly get a large set of live elements}
    {Add E_g to the set of lived elements.}
    $E_{live} = E_{live} \cup E_g$
  end if
until $E_{SPL} = (E_{live} \cup E_{dead})$

Algorithm 2 checks whether elements of E_{SPL} are common or not.

Input: SPL
Output: E_{Common}, the set of common elements of SPL
E_{common} =Ø
{We add the E_{common} all the mandatory elements from each perspective.}
$E_{common} += E_{mandatory}$
{We propagate the truth values of the current common elements among the rest of the constraints...}
$C_{SPL} += E_{common}$
{...and simplify the constraint as much as possible}
Simplify(C_{SPL})
{Simplification leads to obvious constraints such as $\bigwedge e$ meaning that the elements e are mandatory.}
for all New statement $\bigwedge e \in C_{SPL}$ **do**
  $E_{common} = E_{common} \cup \{e\}$
end for
{But all the mandatory elements cannot be found like that, exhaustive search is yet required.}
for all $e \in E_{SPL} \setminus E_{common}$ **do**
  if $C_{SPL} \cup \{\neg e\}$ is not satisfiable **then**
    {e is a common element.}
    $E_{common} = E_{common} \cup \{e\}$
  end if
end for

6. ECPL IN THE CONSTRAINT BASED FRAMEWORK

In this section, we model the ECPL according to the constraint based framework defined in Section 4.

6.1 Constraints of the feature perspective

As defined earlier, the elements of the feature perspective are the propositional symbols of the features and the input and output variables. The set of constraints corresponding to the Czarnecki feature model can be derived as follows :

- p and q are related through a mandatory relation iff $p \Leftrightarrow q \in C_F$.
- p and q are related through an optional relation iff $q \Rightarrow p \in C_F$.
- p and $\langle q_1, \ldots q_p \rangle$ are related through an alternative relation iff $p \Rightarrow q_1 \oplus \cdots \oplus q_i \oplus \cdots \oplus q_n \in C_F$.
- p and $\langle q_1, \ldots q_p \rangle$ are related through an *or* relation with cardinality (n,m) iff $p \Rightarrow \bigvee_{S \subseteq \{q_1 \cdots, q_p\}} (\bigwedge_{s \in S} s), n \leq Card(S) \leq m$.

The variability constraints derived from the Figure 2 are given in the following:

1 $p_{Entry\,control}$

2 $p_{Entry\,control} \Leftrightarrow p_{Power\,lock} \wedge p_{Transmission} \wedge p_{Location}$

3 $p_{Transmission} \Leftrightarrow p_{Manual} \oplus p_{Automatic}$

4 $p_{Location} \Leftrightarrow p_{USA} \oplus p_{Europe}$

5 $p_{Child\,lock} \Rightarrow p_{Entry\,control}$

6 $p_{Theftlock} \Rightarrow p_{Entry\,control}$

7 $p_{Power\,lock} \Leftrightarrow p_{Post\,crash\,unlock}$

8 $p_{Door\,lock} \Leftrightarrow p_{Power\,lock}$

9 $p_{Door\,unlock} \Leftrightarrow p_{Power\,lock}$

10 $p_{Door\,lock} \Leftrightarrow p_{Trigger\,1}$

11 $p_{Trigger\,1} \Rightarrow p_{Shift\,out\,of\,park} \oplus p_{Speed}$

12 $p_{Door\,relock} \Rightarrow p_{Door\,lock}$

13 $p_{Door\,unlock} \Leftrightarrow p_{Trigger\,2} \wedge p_{Door}$

14 $p_{Trigger\,2} \Rightarrow p_{Shift\,into\,park} \oplus p_{Key\,removed}$

15 $p_{Door} \Rightarrow p_{Driver} \oplus p_{All} \oplus p_{No}$

From the requirement document of ECPL, we have extracted the following additional constraints. Constraints of these types cannot be accommodated cleanly in the Czarnecki feature model.

16 $p_{USA} \Rightarrow p_{Door\,lock}$

17 $p_{USA} \Rightarrow p_{Door\,unlock}$

18 $p_{Manual} \Rightarrow p_{Speed}$

19 $p_{Manual} \Rightarrow p_{KeyRemoved}$

6.2 Constraints of the architecture perspective

The elements of the architecture perspective are the components (including the root component) and the links. The set of constraints corresponding to the *Modal Architecture Model* can be derived as follows: Let $\ell = (p_1, p_2) \in L$ be an interconnection and let p_i be a port of the component c_i, $i \in \{1, 2\}$. We have

- If a component c is mandatory, then $p_c \in C_A$
- If an interconnection ℓ is mandatory, then $\ell \Leftrightarrow c_1 \wedge c_2 \in C_A$,
- If an interconnection ℓ is optional, then $\ell \Rightarrow c_1 \wedge c_2 \in C_A$.

From Figure 3, we have extracted the following four constraints (ABD stands for *Airbag deployed*).

20 $p_{Link\,ABD\,Child\,lock\,cmp} \Rightarrow p_{Child\,lock\,cmp}$

21 $p_{Link\,ABD\,Theft\,lock\,cmp} \Rightarrow p_{Theft\,lock\,cmp}$

22 $p_{Link\,ABD\,Auto\,lock\,cmp\,Child\,lock\,cmp} \Rightarrow p_{Auto\,lock\,cmp} \wedge p_{Child\,lock\,cmp}$

23 $p_{Link\,ABD\,Auto\,lock\,cmp\,Theft\,lock\,cmp} \Rightarrow p_{Auto\,lock\,cmp} \wedge p_{Theft\,lock\,cmp}$

From the requirement document of ECPL, we had extracted the following four additional constraints. Again, it is not easy to capture these constraints in the pictorial *Modal Architecture Model*.

Since the $Child\ lock$ feature is implemented using the two components $Child\ lock$ and $Child\ lock\ LED$, The relation 24 should be in C_A.

24 $p_{Child\,lock\,cmp} \Leftrightarrow p_{Child\,lock\,LED\,cmp}$

The propagation of the *Airbag deployed* input can be implemented by two different ways: each of the following components: $Auto\ lock$, $Child\ lock$, and $Theft\ lock$ can have either a direct connection from the global "in" port or only $Auto\ lock$ has a direct connection and it propagates this signal to the two other components.

25 $p_{Link\,ABD\,Child\,lock\,cmp} \Leftrightarrow p_{Link\,ABD\,Theft\,lock\,cmp}$

26 $p_{Link\,ABD\,Auto\,lock\,cmp\,Child\,lock\,cmp} \Leftrightarrow p_{Link\,ABD\,Auto\,lock\,cmp\,Theft\,lock\,cmp}$

27 $p_{Link\,ABD\,Child\,lock\,cmp} \Leftrightarrow \neg p_{Link\,ABD\,Auto\,lock\,cmp\,Child\,lock\,cmp}$

6.3 Constraints of the behavior perspective

The elements of the behavior perspective are the transitions. The variability constraint corresponding to the automata of Section 2 can be extracted as follows: Let $vp = \{t_1, ...t_i, ...t_m\}$ be a set of transitions crossed by a bent arrow. Then $vp \Rightarrow t_1 \oplus ... \oplus t_i \oplus ... \oplus \mathcal{T}_m \in C_B$. If this variation point has to be considered, only one transition can be present in any product.

From the figures 4, 5, and 6, we extract the following constraints:

28 $p_{Door\,lock\,trigger} \Rightarrow p_{Speed} \oplus p_{Shift\,out\,of\,park}$ (from Figure 4)

29 $p_{Door\,unlock\,trigger} \Rightarrow p_{Key\,removed} \oplus p_{Shift\,into\,park}$ (from Figure 5)

30 $p_{Door\,unlock\,door\_1} \Rightarrow p_{Driver} \oplus p_{All} \oplus p_{No}$ (from Figure 5)

31 $p_{Door\,unlock\,door\_2} \Rightarrow p_{Driver} \oplus p_{All} \oplus p_{No}$ (from Figure 5)

32 $p_{Pcu_1} \Rightarrow p_{Unlock\,nothing} \oplus p_{Unlock\,child\,lock} \oplus p_{Unlock\,theft\,lock} \oplus p_{Unlock\,both}$ (from Figure 6)

33 $p_{Pcu_2} \Rightarrow p_{Unlock\,theft\,lock\,first} \oplus p_{Unlock\,child\,lock\,first} \oplus p_{Unlock\,both\,in\,parallel}$ (from Figure 6)

Let us add an extra requirement specifying that if both Child Lock and Theft Lock are present, then Child lock should be deactivated first. This gives us an additional constraint,

34 $p_{Pcu_2} \Leftrightarrow p_{Unlock\,child\,lock\,first}$.

6.4 Traceability Constraints

Traceability from features to architecture. According to Table 2, the traceability constraints are the following:

35 $p_{Power\ lock} \Rightarrow p_{Power\ lock\ cmp} \wedge p_{Door\ lock\ manager\ cmp}$

36 $p_{Child\ lock} \Rightarrow p_{Child\ lock\ cmp} \wedge p_{Child\ lock\ LED\ cmp}$

37 $p_{Theft\ lock} \Rightarrow p_{Theft\ lock\ cmp}$

38 $p_{Post\ crash\ unlock} \Rightarrow p_{Auto\ lock\ cmp}$

39 $p_{Door\ look} \Rightarrow p_{Auto\ lock\ cmp}$

40 $p_{Door\ unlock} \Rightarrow p_{Auto\ lock\ cmp}$

41 $p_{Door\ relock} \Rightarrow p_{Auto\ lock\ cmp}$

In addition, we have the constraints relating the input/output signals of the features to the in-ports and out-ports of the root component.

Traceability from features to behaviors. Because of correlation between the variability in the feature perspective and the behavior perspective, the following requirements has to be translated in constraints

- The choice of the doors (driver, all, no) referred by the Door unlock feature is the same in every models.

  42 $p_{Door} \Leftrightarrow p_{Door\ unlock\ door\_1}$

  43 $p_{Door} \Leftrightarrow p_{Door\ unlock\ door\_2}$

- The selection of the trigger of the feature Door lock (resp. Door unlock) is the same in every model.

  44 $p_{Trigger\ 1} \Leftrightarrow p_{Door\ lock\ trigger}$

  45 $p_{Trigger\ 2} \Leftrightarrow p_{Door\ unlock\ trigger}$

- Concerning Post crash unlock, if child lock feature is enabled, it is opened first. Then if theft lock feature is enabled, it is opened before unlocking the doors.

  46 $p_{Unlock\ nothing} \Leftrightarrow \neg p_{Child\ lock} \wedge \neg p_{Theft\ lock}$

  47 $p_{Unlock\ child\ lock} \Leftrightarrow p_{Child\ lock} \wedge \neg p_{Theft\ lock}$

  48 $p_{Unlock\ theft\ lock} \Leftrightarrow \neg p_{Child\ lock} \wedge p_{Theft\ lock}$

  49 $p_{Unlock\ both} \Leftrightarrow p_{Child\ lock} \wedge p_{Theft\ lock}$

6.5 Consistency, liveness, and commonness checking

For the ECPL, the number of feature products is 640, number of architecture products is 32 and the number of variations within the behavior perspective is 72. Taken separately, these give rise to 640x32x72=1474560 combinations. However, with traceability constraints, the number of products turns out to be only 460. We verified that all the elements are live (each element is used in at least one product) except for the transitions $Unlock\ theft\ lock\ first$ and $Unlock\ both\ in\ parallel$ which are dead due to the constraint 34. We also verified that features $Power\ lock$ and $Post\ crash\ unlock$ are common to all the products. Also, since the feature $Post\ crash\ unlock$ is mandatory, the features $Door\ lock$, $Door\ unlock$, and $Door\ relock$ are also common because they are all implemented by the $Auto\ lock$ component. The $Auto\ lock$ component is also common.

7. CONCLUSION

In this paper, we have proposed a constraint based framework for SPL. We claim that the framework provides a uniform and expressive formalism to describe the constraints within and across perspectives. Thus, the analysis problems of SPLs can be addressed in a framework which integrates all the perspectives of a given SPL. We have illustrated this approach through a case study of entry control system for automobiles. The integrated set of constraints comprises both intra- and inter- perspective constraints (variability and traceability). Our experiments show that the set of constraints modeling the entire SPL can be efficiently analyzed for consistency and computing live and common elements.

The large number of possible products of an SPL is an issue in the industry. This has an implication on the effort spent to derive the desired products. We have suggested that traceability constraints capturing some aspects of design (of both architecture and algorithms in the components) can reduce this number by a large extent. The case study provides some evidence to this effect.

The number of constraints can become a problem in case of very large SPLs because all the constraints of the SPLs are integrated in the same constraint solving framework. However, the specific structure of constraints (most constraints are implications) can be exploited to optimize the analysis methods. Domain and application specific study should be carried out to identify specific optimization techniques.

8. REFERENCES

[1] E. Adachi, T. Batista, U. Kulesza, A. L. Medeiros, C. Chavez and A. Garcia. Variability management in aspect-oriented architecture description languages: An integrated approach. *Brazilian Symposium on Software Engineering*, 2009.

[2] D. S. Batory. Feature models, grammars and propositional formulas. In J. H. Obbink and K. Pohl, editors, *SPLC*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[3] D. Beuche. Modeling and building software product lines with pure: : Variants. In *SPLC*, page 358, 2008.

[4] J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. ACM Press/ Addison - Wesley Publishing Co., New York, NY, USA, 2000.

[5] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Tools and Applications*. ACM Press, Addison-Wesley Publishing Co. New York, NY, USA, June 2000.

[6] E. A. de Oliveira, Junior, I. M. S. Gimenes, E. H. M. Huzita and J. C. Maldonado. A variability management process for software product lines. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 225–241. IBM Press, 2005.

[7] D. Dhungana and P. Grünbacher. Understanding decision-oriented variability modelling. In Thiel and Pohl [29], pages 233–242.

[8] A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *SPLC*, pages 193–202, 2008.

[9] P. C. C. Felix Bachmann. Variability in software product lines. Technical Report TR-012, CMU/SEI, 2005.

[10] D. Garlan, R. T. Monroe and D. Wile. Acme: an architecture description interchange language. In J. H. Johnson, editor, *CASCON*, page 7. IBM, 1997.

[11] H. Gomaa and D. L. Webber. Modeling adaptive and evolvable software product lines using the variation point model. In *HICSS '04: Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.

[12] A. Gruler, M. Leucker and K. D. Scheidemann. Calculating and modeling common parts of software product lines. In *SPLC*, pages 203–212, 2008.

[13] A. V. D. Hoek. Capturing product line architectures. In *In Proceedings of the 4th International Software Architecture Workshop*, number CU-CS-895-99, pages 2000–95, 2000.

[14] M. Janota, J. Kiniry and G. Botterweck. Formal methods in software product lines: Concepts, survey and guidelines, 2008.

[15] S. Krishnamurthi and K. Fisler. Foundations of incremental aspect model-checking. *ACM Trans. Softw. Eng. Methodol.*, 16(2):39, 2007.

[16] C. W. Krueger and K. Jackson. Requirements engineering for systems and software product lines, 2009.

[17] R. Laney, T. T. Tun, M. Jackson and B. Nuseibeh. Composing features by managing inconsistent requirements. In L. du Bousquet and J.-L. Richier, editors, *Ninth International Conference on Feature Interactions in Software and Communication Systems (ICFI'07)*, pages 141–156, 2007.

[18] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens and G. Saval. Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 243–253, 2007.

[19] U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Department of computer science, Aalborg University, Denmark, 2008.

[20] S. E. I. of Carnegie Mellon University. Software product line web site: http://www.sei.cmu.edu/productlines, 2010.

[21] G. Perrouin, F. Chauvel, J. DeAntoni and J.-M. Jézéquel. Modeling the variability space of self-adaptive applications. In Thiel and Pohl [29], pages 15–22.

[22] K. Pohl, G. Böckle and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[23] M. Riebisch and R. Brcina. Optimizing design for variability using traceability links. In *ECBS'08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 235–244, Washington, DC, USA, 2008. IEEE Computer Society.

[24] F. Roos-Frantz. A preliminary comparison of formal properties on orthogonal variability model and feature models. In D. Benavides, A. Metzger and U. W. Eisenecker, editors, *VaMoS*, volume 29 of *ICB Research Report*, pages 121–126. Universität Duisburg-Essen, 2009.

[25] T. K. Satyananda, D. Lee and S. Kang. Formal verification of consistency between feature model and software architecture in software product line. In *ICSEA '07: Proceedings of the International Conference on Software Engineering Advances* [26], page 10.

[26] T. K. Satyananda, D. Lee, S. Kang and S. I. Hashmi. Identifying traceability between feature model and software architecture in software product line using formal concept analysis. *Computational Science and its Applications, International Conference*, 0:380–388, 2007.

[27] L. Shen, X. Peng and W. Zhao. A comprehensive feature-oriented traceability model for software product line development. In *Software Engineering Conference, 2009. ASWEC '09. Australian*, pages 210 –219, April 2009.

[28] M. Svahnberg, J. van Gurp and J. Bosch. A taxonomy of variability realization techniques: Research articles. *Softw. Pract. Exper.*, 35(8):705–754, 2005.

[29] S. Thiel and K. Pohl, editors. *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*. Lero Int. Science Centre, University of Limerick, Ireland, 2008.

[30] M. Tommi. Variability management in software product-lines. Technical Report 30, Institute of Software Systems, Tampere University of Technology, January 2002.

[31] R. C. van Ommering, F. van der Linden, J. Kramer and J. Magee. The koala component model for consumer electronics software. *IEEE Computer*, 33(3):78–85, 2000.

[32] M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *SPLC '07: Proceedings of the 11th International Software Product Line Conference*, pages 233–242, Washington, DC, USA, 2007. IEEE Computer Society.

[33] Website. http://www.win.tue.nl/ wieger/bddsolve/.

[34] Website. http://yices.csl.sri.com/.

[35] J. White, D. C. Schmidt, D. Benavides, P. Trinidad and A. Ruiz-Cortés. Automated diagnosis of product-line configuration errors in feature models. In *Proceedings of the 12th International Software Product Line Conference*, pages 225–234, Washington, DC, USA, 2008. IEEE Computer Society.

[36] C. Zhu, Y. Lee, W. Zhao and J. Zhang. A feature oriented approach to mapping from domain requirements to product line architecture. In H. R. Arabnia and H. Reza, editors, *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006, Las Vegas, Nevada, USA, June 26-29, Volume 1*, pages 219–225. CSREA Press, 2006.

APPENDIX

A. MODAL ARCHITECTURE MODEL

DEFINITION 8 (MODAL ARCHITECTURE MODEL). *A modal architecture model (MAM) is a tuple* $\langle C, c_{spl}, Link, \Delta_c, \Delta_l \rangle$ *where:*

- C *is a set of components.*
- c_{spl} *is the root component.*
- $Link$ *is a set of links or interconnections* (p_1, p_2) *where,* p_1 *(resp.* p_2*) is an* out *(*in*) port of a component* c_1 *(*c_2*).*
- $\Delta_c : C \rightarrow \{may, must\}$ *associates a modality to a component.*
- $\Delta_l : Link \rightarrow \{may, must\}$ *associates a modality to a link.*