Combining Variability, RCA and Feature Model for Context-Awareness

Anne Marie Amja, Abdel Obaid, Hafedh Mili
LATECE Research Lab.
Department of Computer Science
University of Quebec at Montreal
Montreal, Canada
anne.amja@gmail.com, obaid.abdel@gmail.com, mili.hafedh@uqam.ca

***Abstract*—As the notion of context-awareness evolves in different paradigms, the development of context-aware systems involves several processes. These processes include, inter alia, context modeling and reasoning as well as adaptation. In [1], we presented an approach that consists of context modeling and reasoning hand in hand based on relational concept analysis and descriptive logic, respectively. An essential aspect that is often neglected in context modeling and also triggers adaptation is the context variability. In this paper, we propose an approach that lies to answer this matter based on software product line. Our approach creates a semantic link between a context RCA-based model and a feature model, and uses the MAPE-K adaptation loop to determine the appropriate SPL configurations to deploy with regards to context changes. Thus, we used ontology to represent a combined context and feature model. Thereafter, the reasoning is done via descriptive logic. We also defined context rules based on SWRL and applied them to valid SPL configurations. Furthermore, we implemented the MAPE-K adaptation loop with Prolog.**

Keywords*—*variability; features; context; rules; software product line; adaptation; control loop

I. Introduction

Context-aware systems demand the ability to self-adapt dynamically and autonomously when contextual changes are detected either internally or externally in order to continue to meet its objectives. Adaptation loops are used by context-aware systems to change their behaviour according to context. A well-recognized reference model to realize adaptation is by means of a control loop called MAPE-K and conceived as a sequence of four phases *Monitor-Analyze-Plan-Execute* over a *Knowledge base*.

Several general architectures are given in the literature to design context-aware systems and usually relate to three phases of *sensing* (e.g. data acquisition and retrieval), *thinking* (e.g. context modeling and reasoning) and *acting* (e.g. planning and executing actions).

Among the various methods available for context modeling, the concept of context variability is often omitted. Context variability is strongly related to the dynamic adaptation of a system due to context changes and depends on the selected context modeling method. We define context variability as a range or a set of context values of an environment along which that specific environment changes or is influenced by various context entities.

Variability modeling used commonly to analyze and represent commonalities and variabilities among software product lines can serve as a benefit for context-aware systems regarding context variability and adaptation.

In our previous work [1], we presented an approach that consists of modeling context based on relational concept analysis and reasoning on the context with description logic. This paper proposes an approach that 1) creates a relationship between a context RCA-based model and a feature model in order to describe the variability of contexts in which software of a software product line is implemented, and 2) employs the MAPE-K reference model to specify the features of a valid software product line configuration that must be activated or deactivated according to context changes.

Our objectives in this work are four-fold: 1) establish a semantic relationship between context-based services and features of a variability model, 2) apply contextual rule-based situation to valid SPL configurations, 3) reason on the context rules and SPL configurations with respect to a context model and a semantic variability model, respectively, and 4) employ an adaptation loop to determine the SPL configuration that needs to be run for a given context.

The remainder of this paper is organized as follows. Related work is summarized in Section 2. Section 3 presents preliminaries of a semantic feature model using ontology. In Section 4, we present our approach for establishing a semantic relationship between context and features in which context variability is taken into account, and for using MAPE-K to determine valid SPL configurations to employ given a context. Section 5 presents the relationship between a context RCA-based model and a variability model. We discuss a rule-based reasoning approach using Semantic Web Rule Language for context-aware systems in section 6. In Section 7, we provide details on the implementation of the MAPE-K control loop based on Prolog. Section 8 concludes this paper and provides a summary of future work.

II. Related Work

In this section, we show the related work from the viewpoint of feature model used in software product line for context-aware applications to relate features and contexts as well as to apply adaptation of such systems.

In [4], Hartmann et al. introduced the concept of context variability model and defined dependencies (i.e. requires, excludes and cardinality) between a context variability model and a feature model. Capilla et al. [5] proposed a context variability approach that consists of modeling context-features in 2 different strategies. The first strategy models context-features separately from non-context features whereas the second strategy models context and non-context features under the same feature model. Tun et al. [6] used contextual variations as links between requirement variations and feature variations. Several feature models are used to separate system descriptions related to requirements, problem-world context and software specifications. In [7], Salifu et al. proposed an approach to variability of software product families that deal with context-awareness by linking requirements to software architecture.

All of the aforementioned approaches model contextual variations separately from feature model and relate both models through dependencies such as *requires* and *excludes*.

In [8], Alferez et al. proposed a framework that uses variability models at run time to support the dynamic adaptation of service composition. Their framework spans over design time and runtime, and states the models and tools used to support dynamic adaptation. Cetina et al. [9] developed the Model-Based Reconfiguration Engine (MoRE) that leverages variability model at runtime to achieve autonomic computing. MoRE implements model-management operations that consist of determining how a system should evolve and the mechanisms for modifying a system architecture. In [10], Parra et al. proposed a Context-Aware Dynamic Service-Oriented Product Line name CAPucine for building service-oriented applications and adapting them at runtime in accordance with their context. It is based on two processes. The first process consists of transforming models (i.e. features of a product family) to generate the product whereas the second process relates to dynamic adaptation based on COSMOS and FraSCAti platforms. Saller et al. [11] proposed an approach for modeling context-aware dynamic software product line adaptation scenarios for resource-constrained systems. Their approach is based on a feature model enriched with context information and a trade-off between precomputation of reconfiguration scenarios at development time and on-demand evolution at runtime.

Moreover, the above-mentioned approaches lack validation of software product line configuration with respect the feature model caused by dynamic adaptation.

III. Feature Model as Ontology

A. Feature Model

Feature modeling is a technique for capturing commonality and variability in software product lines. In a feature model, there is only one *root* feature that identifies the SPL and beneath it are features hierarchically organized and classified as *mandatory* (i.e. features that are required) or *optional* (i.e. features that are optional).

Relationship between a parent feature and its children features are *and*, *or* or *xor*-relation. Features in an *And*-relation can be either mandatory or optional subfeatures of the parent features. *Or*-relation indicates at least one child feature must be present when its parent is present. *Xor*-relation indicates exactly one child feature must be present if its parent feature is present.

Feature model allows cross-tree constraints: *requires* (i.e. a given feature *requires* the existence of another feature) and *excludes* (i.e. the presence of a given feature *excludes* the elimination of another feature).

In Fig. 1, we illustrate an example of a feature model for a simplified health monitoring system.

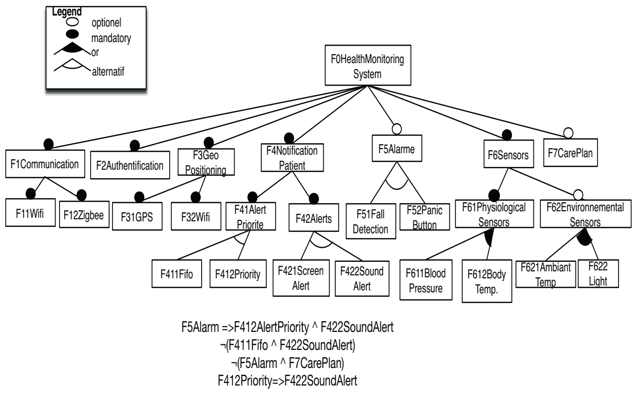


Fig. 1. Feature Model of a Health Monitoring System

B. Feature Model as OWL

Feature models lack of formal semantics and reasoning support. [12]–[14] proposed an approach that consists of modeling a feature model using ontology and verifying them for any inconsistencies with a reasoner.

Based on their work, we've used OWL-DL to capture the inter-relationships among the features in a feature model and to derive the configurations.

The ontology of a feature model is constructed in 3 steps:

a) Identify the nodes in a feature model: consists of modeling each feature as an OWL class make them mutually disjoint.

b) Create object-property: consists of creating an object property for each edge type representing feature relations. The range of the property is the respective feature class.

c) Create Rule classes: consists of creating a Rule class for each feature. The Rule class has 2 conditions:

- A necessary and sufficient condition using an existential restriction to bind the Rule class to the corresponding feature class.
- A number of necessary constraints specifying how each of its child features is related to this class (i.e. the group-relation between parent and children and specifying how a feature node is constrained by other features (i.e. the cross-tree *requires* and *excludes* constraints).

For a parent feature G and its child features $F_1, \ldots, F_n$, the initial modeling summarized above produces the following ontology illustrated by Table I [12]–[14].

TABLE I. DESCRIPTION LOGIC BASE STRUCTURE OF FEATURE MODEL

Feature relation and constraints	**Description logic modeling**
$G \sqsubseteq T$ and $GRule \sqsubseteq T$ $F_i \sqsubseteq T$ and $F_iRule \sqsubseteq T \ldots F_n \sqsubseteq T$ and $F_nRule \sqsubseteq T$	$hasG \sqsubseteq ObjectProperty$, $T \sqsubseteq \forall hasG.G$ and $GRule \equiv \exists hasG.G$
$G \sqsubseteq \neg F_i$ for $1 \leq i \leq n$ $F_i \sqsubseteq \neg F_j$ for $1 \leq i, j \leq n$ where $i \neq j$	$hasF_i \sqsubseteq ObjectProperty$, $T \sqsubseteq \forall\ hasF_i .F_i$, $F_iRule \equiv \exists\ has.F_i.F_i$ for $1 \leq i \leq n$

The following relations from the feature model are modeled in OWL-DL and summarized in Table II [12]–[14].

a) Mandatory features: For each mandatory features $F_1, \ldots, F_n$ of a parent feature G, a *someValuesFrom* restriction is used (i.e. each instance of the class GRule must have some instance of F_i class for $hasF_i.GRule \sqsubseteq \exists hasF_1.F_1 \ldots GRule \sqsubseteq \exists\ hasF_n.F_n$).

b) Optional features: For each optional features $F_1, \ldots, F_n$ of a parent feature G, the Rule class of each child feature class is made as a subclass of the *someValuesFrom* restriction on the parent feature class.

c) Alternative features: For a set of *alternative* features $F_1, \ldots, F_n$ and a parent feature G, a disjunction of *someValuesFrom* restrictions over $hasF_i$ is used to ensure that some feature will be included.

d) Or features: For a set of *or* features $F_1, \ldots, F_n$ of a parent G, a disjunction of *someValuesFrom* restrictions is used to model this relation and omit the negation of conjunctions to allow multiple or features to be included in the configuration.

e) Requires: For a given feature G and a set of features $F_1, \ldots, F_n$ that G includes, besides the necessary and sufficient (NS) condition that binds GRule to G, each of the F_i features appears in a configuration if G is present.

f) Excludes: For a given feature G and a set of features $F_1, \ldots, F_n$ that G excludes, a negation of *someValuesFrom* restriction on $hasF_i$ property is used to ensure that GRule doesn't have any F_i feature.

TABLE II. DESCRIPTION LOGIC REPRESENTATION OF FEATURE MODEL

Feature type and constraints	**Description logic modeling**
Mandatory	$GRule \equiv \exists\ has.F_1.F_1, \ldots, GRule \equiv \exists\ has.F_n.F_n$
Optional	No constraints
Or	$GRule \sqsubseteq \sqcup(\exists\ has.F_i.F_i)$ for $1 \leq i \leq n$
Alternative	$GRule \sqsubseteq \sqcup(\exists\ has.F_i.F_i)$ for $1 \leq i \leq n$ $GRule \sqsubseteq \neg\sqcup(\exists\ has.F_i.F_i \sqcap \exists\ has.F_i.F_i)$ for $1 \leq i \leq j \leq n$
Requires	$GRule \equiv \exists\ has.F_i.F_i, \ldots, GRule \equiv \exists\ has.F_n.F_n$
Excludes	$GRule \equiv \neg(\exists\ has.F_i.F_i), \ldots, GRule \equiv \neg(\exists\ has.F_n.F_n)$

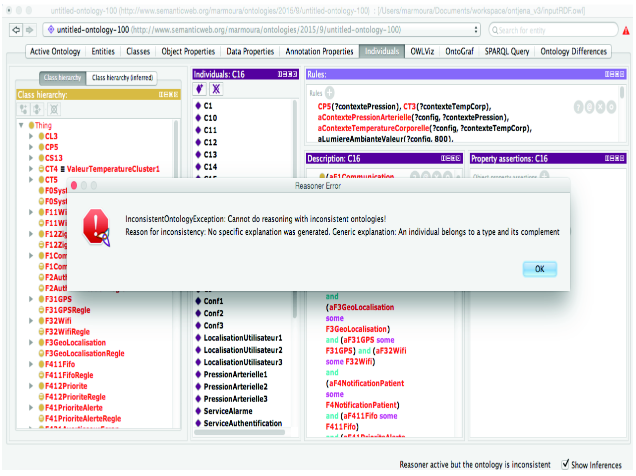
C. *Software Product Line in OWL*

Based on our Health Monitoring System example, we've used the Protégé ontology editor [15] to model the feature model and Pellet reasoner [16] to verify for any inconsistencies of SPL configurations.

Suppose that we have a configuration C16 that consists of an instance of the class F0HealthMonitoringSystemRule (Fig. 2).

C16 ≡(hasF1Communication some F1Communication) ⊓ (hasF11Wifi some F11Wifi) ⊓ (hasF12Zigbee some F12Zigbee) ⊓ (hasF2Authentification some F2Authentification) ⊓ (hasF3GeoPositioning some F3GeoPositioning) ⊓ (hasF31GPS some F31GPS) ⊓ (hasF32Wifi some F32Wifi) ⊓ (hasF4PatientNotification some F4PatientNotification) ⊓ (hasF41AlertPriority some F41AlertPriority) ⊓ (not(hasF411FIFO some F411FIFO)) ⊓ (not(hasF412Priority some F412Priority)) ⊓ (hasF42Alerts some F42Alerts) ⊓ (not(hasF421ScreenAlert some F421ScreenAlert)) ⊓ (not(hasF422SoundAlert some F422SoundAlert)) ⊓ (hasF5Alarm some F5Alarm) ⊓ (hasF51FallDetection some F51FallDetection) ⊓ (hasF6Sensors some F6Sensors) ⊓ (hasF61PhysiologicalSensor some F61PhysiologicalSensor) ⊓ (hasF611BloodPressure some F611BloodPressure) ⊓ (hasF62EnvironmentalSensor some F62EnvironmentalSensor) ⊓ (hasF621AmbiantTemp some F621AmbiantTemp) ⊓ (hasF7CarePlan some F7CarePlan)

The Pellet reasoner will complain that C16 is inconsistent (see Fig. 2). A closer analysis reveals that F5Alarm *excludes* F7CarePlan, which are both present in the current configuration.



```
CP5(?contextePression), CT3(?contexteTempCorp),
aContextePressionArterielle(?config, ?contextePression),
aContexteTemperatureCorporelle(?config, ?contexteTempCorp),
aLumiereAmbianteValeur(?config, 800),
```

InconsistentOntologyException: Cannot do reasoning with inconsistent ontologies!
Reason for inconsistency: No specific explanation was generated. Generic explanation: An individual belongs to a type and its complement

Reasoner active but the ontology is inconsistent

Fig. 2. Inconsistency of the SPL configuration

We correct the configuration C1 by adding a negation to F7CarePlan but the Pellet reasoner still complains about an inconsistency. The source of inconsistency also comes from F422SoundAlert. We have 2 constraints: 1) F5Alarm *requires* both F412Priority and F422SoundAlert, and 2) F412Priority *requires* F422SoundAlert. Once we remove the negation to F412Priority and F422SoundAlert from the C16 configuration, Pellet reasoner confirms that the ontology is consistent. Thus, the C16 configuration is now valid.

IV. Proposed Approach

In our previous work [1], we proposed a context modelling approach based on relational concept analysis (RCA), an extension of formal concept analysis, with the intention of establishing dependencies between context and services. We also employed existing mapping rules between the source entities of RCA and the target entities of Description Logic to reason on context.

In this work, we propose a novel approach that consists of describing context variability by means of a relationship between context RCA-based model and feature model as well as employing the MAPE-K control loop in order to determine an appropriate SPL configuration (w.r.t the feature model) to deploy for a given a context.

Note that in our approach, we describe the context variability to valid SPL configurations as a whole instead of separate and selected features and we perform adaptation in terms of activation/deactivation of features based on context variability. We illustrate our proposed approach in Fig. 3.

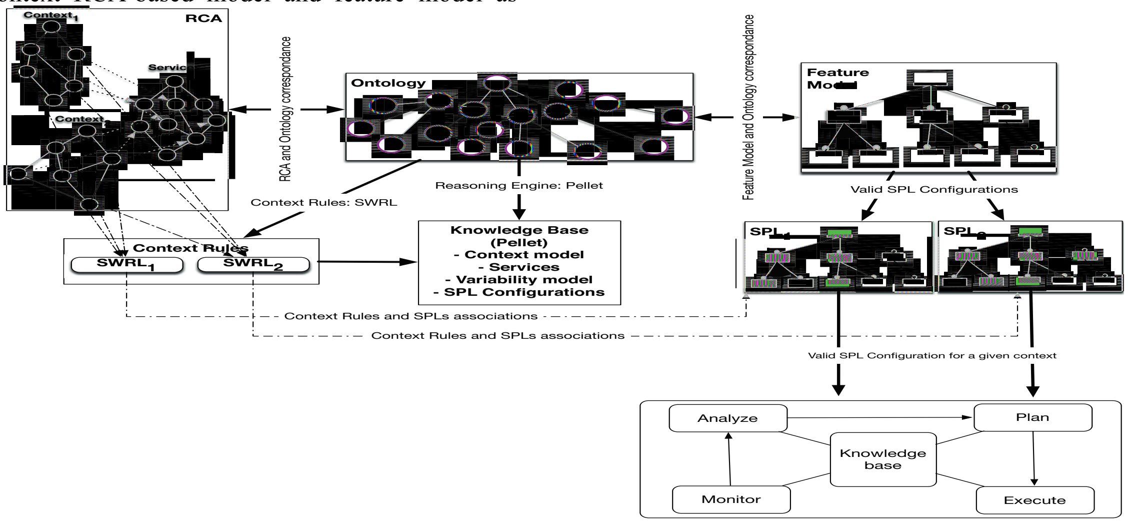


Fig. 3. Proposed approach

It mainly consists of the following modules:

1) Link service lattice to features of a variability model: establishes a relationship between the service context-based lattice and the features of a variability model. We present in section 5 the relationship accomplished through the attributes of service lattice as features of the variability model.

2) Represent a Semantic feature model: consists of using the approach presented in section 3 to model and verify semantically a feature model.

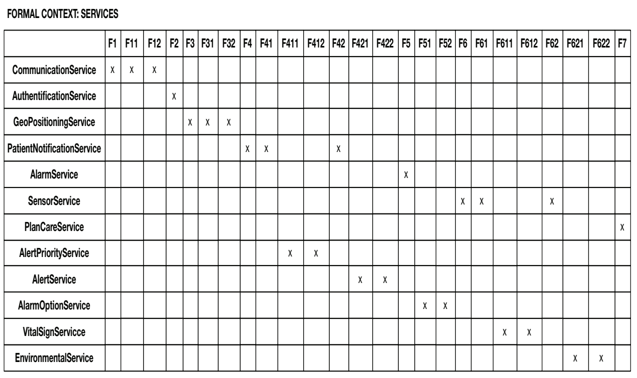
3) Create context rules: creates context rules for SPL configurations. The context rules derive from the context modeling approach and are associated to its corresponding and valid SPL configurations with respect to a variability model of a context-aware system. We use SWRL to represent context rules that are applied to SPL configurations with respect to the feature model. We present our approach in section 6.

4) Reason with ontologies: validates SPL configurations and SWRL context rules along with our corresponding ontology mainly composed of context, services and features by using Pellet [16] and the Protégé ontology editor [15]. Sections 3 and 6 detail the reasoning of SPL configurations and SWRL context rules, respectively.

5) Transit from a current SPL configuration to a target SPL configuration with respect to context changes: consists of using the MAPE-K reference model to transit from one valid SPL configuration to another when the context changes. Each phase *Monitor-Analyze-Plan-Execute* and the interacting *Knowledge base* of the MAPE-K loop are written in Prolog. Section 7 details the execution of the adaptation process.

V. Link Between Context Model and Feature Model

Context modeling with RCA requires two types of lattices: context lattices and service lattice. Context lattices are linked to the service lattice to show dependencies among formal concepts. We establish a relationship between the service lattice and the features of a variability model. To do so, the objects represent *services* whereas the attributes represent *features* of a variability model, respectively, in a formal context. We omit the *root feature*, which represents a system and is considered as an *abstract feature*. In other words, the service lattice derives a concept hierarchy or formal ontology from a collection of services and their properties. These properties are the features found in a feature model. In Fig. 4, we illustrate the main concept. We note that attributes' name by the features' identification number.



FORMAL CONTEXT: SERVICES

	F1	F11	F12	F2	F3	F31	F32	F4	F41	F411	F412	F42	F421	F422	F5	F51	F52	F6	F61	F611	F612	F62	F621	F622	F7
CommunicationService	x	x	x																						
AuthentificationService				x																					
GeoPositioningService					x	x	x																		
PatientNotificationService								x	x			x													
AlarmService															x										
SensorService																		x	x			x			
PlanCareService																									x
AlertPriorityService										x	x														
AlertService													x	x											
AlarmOptionService																x	x								
VitalSignServicce																				x	x				
EnvironmentalService																							x	x	

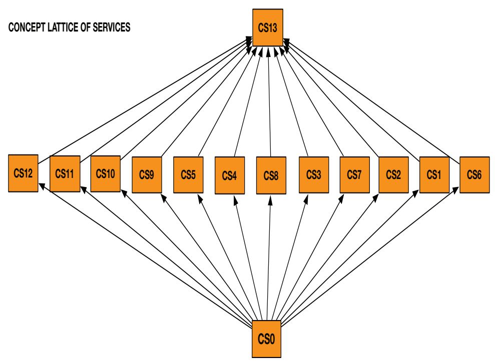


Fig. 4. Service lattice

As proposed in [1], we show in Fig. 5, the final contexts and service lattices obtained when applying RCA for the health monitoring system based on 2 contexts: blood pressure and body temperature. We've used RCAExplore [17] to model the context based on RCA and defined it as an ontology model with the Protégé ontology editor [15].

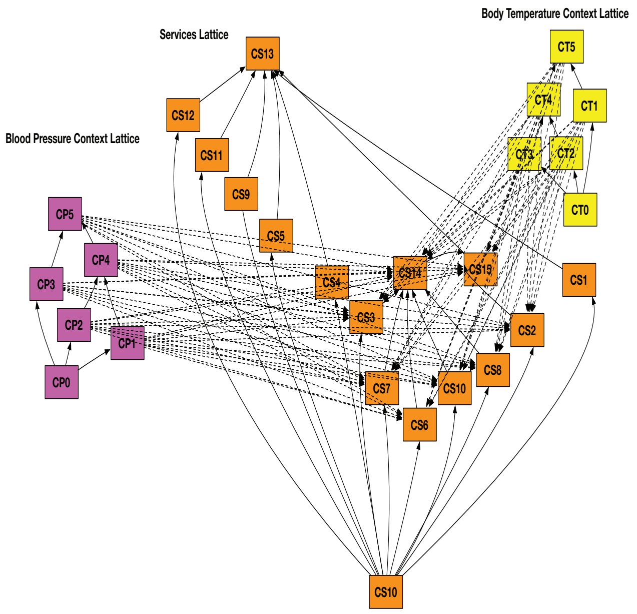


Fig. 5. Context and service lattices of Health Monitoring System

VI. SPL Configuration and Context Rules

Ontologies lack support for free variables that are required to express dependencies on the SPL configurations. Semantic Web Rule Language (SWRL) adds rule capabilities to ontologies. Such rules are in the form of an implication between an antecedent and consequent [18]. Its intended meaning is read as whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold [18].

We propose to use SWRL to represent context rules that are applied to SPL configurations. We show how the execution of the context rules successfully retrieves the SPL configuration that matches these contexts. The antecedent of a SWRL rule is used to represent a conjunction of context atoms and the consequent represent the specific SPL configuration.

Within Table III, various SWRL rules are described which include the SPL configurations required for the following contexts: *healthy state of patient*s, *low blood pressure of patient*s, and *fever and high blood pressure of patient*s.

We've used the SWRL-Tab available in the Protégé ontology editor [15] to develop and test SWRL rules along with our corresponding ontology.

For instance, the SWRL rule for the context of *fever and high blood pressure of patients* is explained as follows:

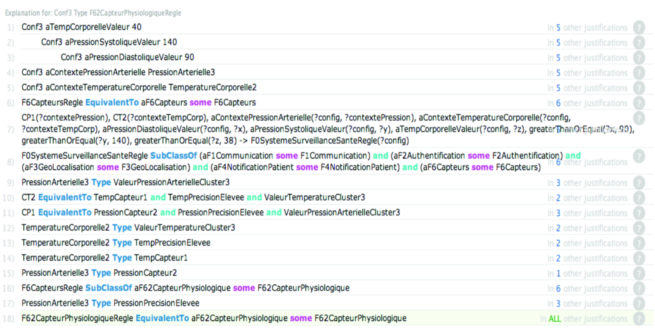
- *CP1(?bloodPressureContext)* constrains the ontology individuals (in this case the SPL configurations) that are applied to the ontology class CP1 by assigning the variable *bloodPressureContext*. *hasBloodPressureContext(?config,?bloodPressureContext)* constrains the ontology individuals which depend on the blood pressure context. *hasDistolicBloodPressureValue(?config,?x)* and *hasSystolicBloodPressureValue(?config,?y)* constrains the ontology individuals with a diastolic blood pressure value greater or equal to 90 *(greaterThanOrEqual(?x,90))* and a systolic blood pressure value greater or equal to 140 *(greaterThanOrEqual(?y,140))*, respectively.
- *CT2(?bodyTemperatureContext)* constrains the ontology individuals (in this case the SPL configurations) that are applied to the ontology class CT2 by assigning the variable *bodyTemperatureContext*. *hasBodyTempContext(?config,?bodyTempContext)* constrains the ontology individuals which depend on the body temperature context. *hasBodyTempValue(?config,?z)* constrains the ontology individuals with a body temperature value greater or equal to 38 *(greaterThanOrEqual(?z,38))*.
- *F0HealthMonitoringSystemRule* indicates the SPL configuration associated to this context rule.

TABLE III. EXAMPLES OF SWRL RULES

Context	SWRL
Healthy state of patients	CP1(?bloodPressureContext), CT2(?bodyTempContext), hasBloodPressureContext(?config, ?bloodPressureContext), hasBodyTempContextCorporelle(?config, ?bodyTempContext), hasDistolicBloodPressureValue(?config, ?x), hasSystolicBloodPressureValue(?config, ?y), hasBodyTempValue(?config, ?z), greaterThanOrEqual(?x, 75), greaterThanOrEqual(?y, 115), greaterThanOrEqual(?z, 36.5), lessThanOrEqual(?x, 80), lessThanOrEqual(?y, 120), lessThanOrEqual(?z, 38) -> F0HealthMonitoringSystemRule (?config)
Low blood pressure of patients	CP1(?bloodPressureContext), hasArteriellePressureContext(?config, ?bloodPressureContext), hasDiastolicBloodPressureValue(?config, ?x), hasSystolicBloodPressureValue(?config, ?y), lessThanOrEqual(?x, 60), lessThanOrEqual(?y, 90) -> F0SystemeSurveillanceSanteRegle(?config)
Fever and high blood pressure of patients	CP1(?bloodPressureContext), CT2(?bobyTempContext), hasBloodPressureContext(?config, ? bloodPressureContext), hasBodyTempContext(?config, ?bobyTempContext),hasDiastolicBloodPressureValue(?config, ?x), hasSystolicBloodPressureValue(?config, ?y), hasBodyTempValue(?config, ?z), greaterThanOrEqual(?x, 90), greaterThanOrEqual(?y, 140), greaterThanOrEqual(?z, 38) -> F0HealthMonitoringSystemRule(?config)

By running Pellet reasoned [16], we obtain the inference explanation of the SPL configuration *conf3* as illustrated in Fig. 6. We explain only the main inference as follows:

- *Conf3* has a body temperature value of 40°C, a systolic and diastolic blood pressure of 140 and 90, respectively.
- *Conf3* depends on the blood pressure context *BloodPressure3* and the body temperature context *BodyTemperature2*.
- The SWRL *fever and high blood pressure* context rule applies to the SPL configuration *Conf3*.
- *BloodPressure3* is of type *CP1*. Blood pressure values of CP1 are sensed by *sensor2* given a high precision and belong to *cluster3*.
- *BodyTemperature2* is of type CP2. Body temperature values of *CT2* are sensed by *sensor1* given a high precision and belong to *cluster3*.



```
Explanation for: Conf3 Type F62CapteurPhysiologiqueRegle
1)  Conf3 aTempCorporelleValeur 40                                                          In 5 other justifications
2)      Conf3 aPressionSystoliqueValeur 140                                                 In 5 other justifications
3)          Conf3 aPressionDiastoliqueValeur 90                                             In 5 other justifications
4)  Conf3 aContextePressionArterielle PressionArterielle3                                   In 5 other justifications
5)  Conf3 aContexteTemperatureCorporelle TemperatureCorporelle2                             In 5 other justifications
6)  F6CapteursRegle EquivalentTo aF6Capteurs some F6Capteurs                                In 6 other justifications
    CP1(?contextePression), CT2(?contexteTempCorp), aContextePressionArterielle(?config, ?contextePression), aContexteTemperatureCorporelle(?config,
7)  ?contexteTempCorp), aPressionDiastoliqueValeur(?config, ?x), aPressionSystoliqueValeur(?config, ?y), aTempCorporelleValeur(?config, ?z), greaterThanOrEqual(?x, 90),
    greaterThanOrEqual(?y, 140), greaterThanOrEqual(?z, 38) -> F0SystemeSurveillanceSanteRegle(?config)
8)  F0SystemeSurveillanceSanteRegle SubClassOf (aF1Communication some F1Communication) and (aF2Authentification some F2Authentification) and
    (aF3GeoLocalisation some F3GeoLocalisation) and (aF4NotificationPatient some F4NotificationPatient) and (aF6Capteurs some F6Capteurs)   In 6 other justifications
9)  PressionArterielle3 Type ValeurPressionArterielleCluster3                               In 3 other justifications
10) CT2 EquivalentTo TempCapteur1 and TempPrecisionElevee and ValeurTemperatureCluster3     In 2 other justifications
11) CP1 EquivalentTo PressionCapteur2 and PressionPrecisionElevee and ValeurPressionArterielleCluster3   In 3 other justifications
12) TemperatureCorporelle2 Type ValeurTemperatureCluster3                                   In 2 other justifications
13) TemperatureCorporelle2 Type TempPrecisionElevee                                         In 2 other justifications
14) TemperatureCorporelle2 Type TempCapteur1                                                In 2 other justifications
15) PressionArterielle3 Type PressionCapteur2                                               In 1 other justifications
16) F6CapteursRegle SubClassOf aF62CapteurPhysiologique some F62CapteurPhysiologique         In 6 other justifications
17) PressionArterielle3 Type PressionPrecisionElevee                                        In 3 other justifications
18) F62CapteurPhysiologiqueRegle EquivalentTo aF62CapteurPhysiologique some F62CapteurPhysiologique   In ALL other justifications
```

Fig. 6. Inconsistency of the SPL configuration

SPL configuration conf3 is represented in Fig. 7. The features in green and white indicate that they are active or inactive, respectively. The SWRL context rule of *fever and high blood pressure of patients* is applied to SPL configuration *conf3*.

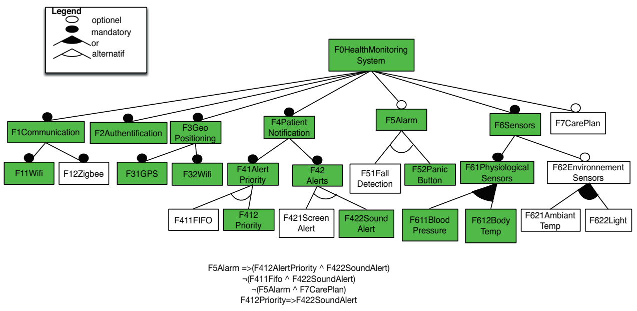


Fig. 7. SPL configuration *conf3*

We also implemented a program in Java and Groovy that reaches two separate objectives: 1) to add or alter SPL configurations of a current OWL file, 2) to validate for any configuration inconsistencies with respect to the variability model by means of the Pellet reasoner [16].

VII. EXECUTION OF THE ADAPTATION PROCESS

We used the MAPE-K [19] control loop to determine the new SPL configuration that will replace the running SPL configuration for a given context rule. Our approach performs adaptation in terms of activation/deactivation of features based on context variability.

Fig. 2 illustrates MAPE-K and we present our implementation with Prolog [20] that plays the role of each MAPE phase (monitoring, analyzing, planning and executing) and the knowledge base.

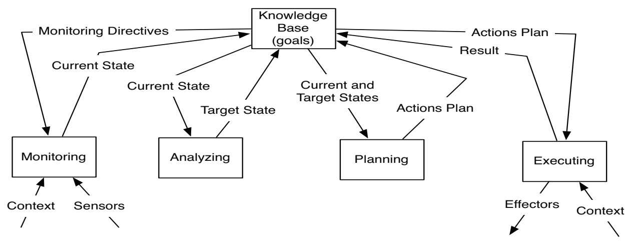


Fig. 8. MAPE-K

A. Monitoring

The knowledge base provides the monitoring directives to the monitor that are represented as contextual properties as follows:

$\{id,\ type,\ delta,\ svalues:\ category{<}e_1{:}i_1, e_2{:}i_2, \ldots, e_n{:}i_n,{>},\ Intervals:\ Category{:}\ {<}e_1{:}i_1, e_2{:}i_2, \ldots, e_n{:}i_n,{>}\}$

where

- *id* identifies the context,
- *type* expresses the type of data associated to the values,

- *delta* represents a value of tolerated error rate in the captured values,
- *svalues: category*$<e_1:i_1,e_2:i_2,\ldots,e_n:i_n>$ indicates the interval of possible values of a category. Category designates the category of data. For instance, in the case of *bodytemperature*, we named the category *oneset*. However, in the case of blood pressure, we'll have 2 categories: *systolic*, which corresponds to the systolic values of the pressure and *diastolic*, which corresponds to the diastolic values of the pressure.
- *Intervals: Category:* $<e_1:i_1,e_2:i_2,\ldots,e_n:i_n>$ is a set of interval values for a contexte. Each interval is associated to a state that identifies it and to which we associate a specific process.

The monitor evaluates the current state of the system and informs the knowledge base.

B. *Analyzing*

The knowledge base informs the analyzer of the current state of the system. Based on the obtained results from the monitor, the analyzer determines the target state to reach from the current state of the system and informs the planner that will define the necessary steps to transit from the current state to the target state. The analyzer also notifies the target state to the knowledge base.

C. *Planning*

We determine the set of features that need to be activated/deactivated to transit from a current state to a target state based on the context. We express SPL configuration as a set of sets of features for a given context.

1) Configuration Expressions and Features

We express the relationships between a parent feature and their children as well as their constraints based on the BNF grammar as follows: E = feature | *and*(E, E) | *or*(E, E) | *xor*(E, E) | *requires*(E, E) | *excludes*(E, E) | *optional*(E). The composition operators constructs a set of features as follows:

- *and(f1, f2)* - both features are required. The set of features is {{f1, f2}}.
- *or(f1, f2)* - both features are mandatory or optional. The sets of features are {{f1}, {f2}, {f1, f2}}.
- *xor(f1, f2)* - both features are mandatory or optional. The sets of features are {{f1}, {f2}}.

2) Expression and set of configurations

We construct a set of features SF for each expression (exp) of a configuration that designates a set of sets of features associated to a feature expression as follows:

- SF(F) ={{F}} if F is a feature.
- If F= *and*(F1, F2)), then S(F)= F.{SF(F1)><SF(F2)}
- If F=*or*(F1,F2), then S(F)= F.{SF(F1),SF(F2), S(F1) >< S(F2)}
- If F= *xor*(F1, F2), then S(F)= F.{SF(F1),SF(F2)}
- If *opt*(F1), then S(F) = F.{{},SF(F)}

Given two sets A and B, the operator >< is defined as follows:

A><B={a ∪ b|a in A, b in B}. Let A ={{},{}}

The operator . adds a feature to every element of a set of features. For a feature f and a set, f.A={<f,a>|a in A}

For example,

- F41=*xor*(F411,F412)

  SF(F41)={{F41,F411},{F41,F412}}
- F42=*xor*(F421,F422)

  SF(F42)={{F42,F421},{F42,F422}}
- F4=*and*(F41,F42)

  SF(F4)={{F4,F41,F411,F42,F421}, {F4,F41,F412,F42,F422}}

Once the SF is found for a given context, the planner informs the knowledge base of its action plan.

D. *Executing*

The knowledge base provides details on the actions plan, defined by the planner, that it has to implement.

1) Configuration Choice

The procedure described above determines the set of configurations that a system can offer. During its execution, a context-aware system may transit from one configuration to another depending on state changes.

In certain cases, these different configurations have features in common. For each state change, we propose one or several configurations that are associated to it. To adapt a system to its changes, we can determine the difference between configurations in the current state and those of the previous one.

2) Differences between configurations: Given two sets of configurations C_A = {af_1, af_2, … , af_n} and C_B = C_A={bf_1, bf_2, … , bf_m}, we construct the following two sets:

- *plus*(C_A, C_B) {b – c, c in C_A , b in C_B} is the set of features that are in a configuration of C_A and not in a configuration of C_B.
- *less*(C_A,C_B) = {c – b, c in C_A , b in C_B} is the set of features that are in a configuration of C_B and not in a configuration of C_A.

For instance, suppose the configuration sets C_A and C_B. Each set is composed of 2 sets of configuration for a given context. C_A is a set of configuration for a normal patient's state where as C_B is a set of configuration for a patient with low blood pressure.

- C_A ={F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F411, F42, F421, F6, F61, F611, F612}, {F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F411, F42, F421, F6, F61, F611, F612, F62, F621, F622, F67}

- C_B={{F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F412, F42, F421, F5, F51, F6, F61, F611, F611, F62, F621}, {F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F412, F42, F421, F5, F51, F6, F61, F611, F611, F62, F621, F622}}

Note, we identify the sets of C_A as C_{A1} and C_{A2} for the first and second set, respectively. We also identify the sets of C_B as C_{B1} and C_{B2} for the first and second set, respectively. *plus*(C_A,C_B) and *less*(C_A,C_B) are summarized in Tables IV and V. The first line of Table indicates that to transit from configuration C_{A1} to C_{B1}, we must add features {F411, F421, F612}. In Table IV, the first line indicates the opposite change, i.e. to transit from configuration C_{B1} to C_{A1}, we must add features {F412, F422, F5, F51, F62, F621}.

TABLE IV. DIFFERENCE OF CONFIGURATION: PLUS(C_A,C_B)

C_A	C_B	***plus*(C_A, C_B)**
C_{A1}	C_{B1}	{F411, F421, F612}
C_{A1}	C_{B2}	{F411, F421, F612}
C_{A2}	C_{B1}	{F411, F421, F612, F622, F7}
C_{A2}	C_{B2}	{F411, F421, F612, F7}

TABLE V. DIFFERENCE OF CONFIGURATION: LESS(C_A,C_B)

C_B	C_A	***less*(C_A, C_B)**
C_{B1}	C_{A1}	{F412, F422, F5, F51, F62, F621}
C_{B2}	C_{A1}	{F412, F422, F5, F51}
C_{B1}	C_{A2}	{F412, F422, F5, F51, F62, F621, F622}
C_{B2}	C_{A2}	{F412, F422, F5, F51}

3) Adaptation graph

The adaptation graph G illustrates these changes of configurations that describes the transition from a step to another in the execution of the system. G = <n_0, N, T> where n_0 is the node of the initial configuration, N is the set of nodes and L is the set of transitions in NxLxN -> N of the form <n_1, l, n_2> where n_1 and n_2 are the nodes (representing a state of a configuration) and l is a function of transition defined by l=*plus*(n_1, n_2)/*less*(n_2, n_1). Fig. 9 presents the adaptation graph.

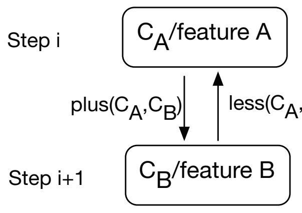


Fig. 9. Adaptation graph

For instance, we obtain the adaptation graph of configurations C_A and C_B illustrated in Fig. 10.

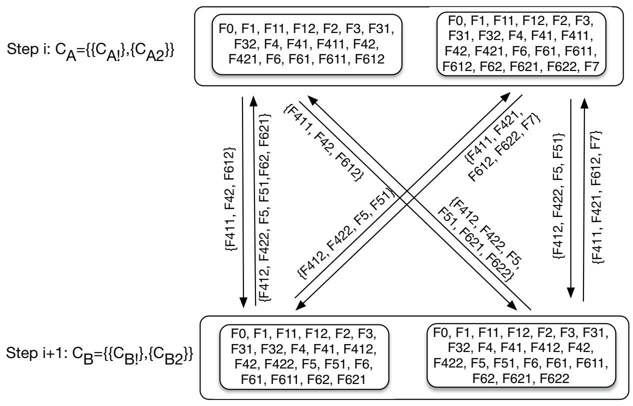

Step i: C_A={{$C_{A!}$},{C_{A2}}}
F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F411, F42, F421, F6, F61, F611, F612
F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F411, F42, F421, F6, F61, F611, F612, F62, F621, F622, F7
{F411, F42, F612}
{F412, F422, F5, F51,F62, F621}
{F411, F42, F612}
{F412, F422, F5, F51}
{F411, F421, F612, F622, F7}
{F412, F422, F5, F51, F621, F622}
{F412, F422, F5, F51}
{F411, F421, F612, F7}
Step i+1: C_B={{$C_{B!}$},{C_{B2}}}
F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F412, F42, F422, F5, F51, F6, F61, F611, F62, F621
F0, F1, F11, F12, F2, F3, F31, F32, F4, F41, F412, F42, F422, F5, F51, F6, F61, F611, F62, F621, F622

Fig. 10. Configuration graph

The adaptation consists of using the adaptation graph at each step and to follow the specified actions in the transitions. In our approach, we use an iterative approach. For each step, we determine the sets of features. Then, for each non-terminal feature belonging to one of the sets, we can deploy it to complete the configuration.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel approach of describing context variability and adaptation of context-aware system. We described context variability to valid software product line configurations instead of selected features by means of a semantic relationship between context RCA-based model and feature model. Thereupon, both semantic models are represented as a single ontology model and the valid SPL configurations are associated to its corresponding context rules based on SWRL. The first part of the implementation was done with the following tools: RCA Explore, Protégé and Pellet. We also employed the MAPE-K control loop to adapt a system based on context changes. The adaptation method consists of specifying the appropriate SPL configuration (i.e. activation/deactivation of features with respect to the feature model) to deploy given a context. The second part of the implementation was developed with Prolog.

As future work, we plan to enhance our proposed model by taking into account the necessity of adding or modifying variants and variation points of a feature model. We also plan to find a plausible solution that will either allow Prolog programs to query and process ontologies or apply a translation rules from ontologies to Prolog.

REFERENCES

[1] A. M. Amja, A. Obaid, and P. Valtchev, "Modeling and reasoning in context-aware systems based on relational concept analysis and description logic," in *2014 IEEE Symposium on Computational Intelligence for Communication Systems and Networks (CIComms)*, 2014, pp. 1–8.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[3] S. Apel, D. Batory, C. Kastner, and G. Saake, *Feature-Oriented Software Product Lines*. Springer Berlin Heidelberg, 2013.

[4] H. Hartmann and T. Trew, "Using feature diagrams with context

variability to model multiple product lines for software supply chains," *Proc. - 12th Int. Softw. Prod. Line Conf. SPLC 2008*, pp. 12–21, 2008.

[5] R. Capilla, O. Ortiz, and M. Hinchey, "Context Variability for Context-Aware Systems," *IEEE Comput.*, vol. 47, no. 2, pp. 85–87, 2014.

[6] T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans, "Relating requirements and feature configurations: a systematic approach," *SPLC - Int. Conf. Softw. Prod. lines*, pp. 201–210, 2009.

[7] M. Salifu, B. Nuseibeh, and L. Rapanotti, "Towards Context-Aware Product-Family Architectures," in *2006 International Workshop on Software Product Management (IWSPM'06 - RE'06 Workshop)*, 2006, pp. 38–43.

[8] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models," *J. Syst. Softw.*, vol. 91, no. 1, pp. 24–47, 2014.

[9] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Autonomic computing through reuse of variability models at runtime: The case of smart homes," *Computer (Long. Beach. Calif).*, vol. 42, no. 10, pp. 37–43, 2009.

[10] C. Parra, X. Blanc, and L. Duchien, "Context awareness for dynamic service-oriented product lines," *13th Int. Softw. Prod. Line Conf.*, pp. 131–140, 2009.

[11] K. Saller, M. Lochau, and I. Reimund, "Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems," *Proc. 17th Int. Softw. Prod. Line Conf. co-located Work.*, pp. 106–113, 2013.

[12] H. H. Wang, Y. F. Li, J. Sun, H. Zhang, and J. Pan, "Verifying feature models using OWL," *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 5, no. 2, pp. 117–129, 2007.

[13] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan, "A semantic web approach to feature modeling and verification," in *1st Workshop on Semantic Web Enabled Software Engineering*, 2005.

[14] M. M. Piash, S. Ripon, and S. A. Hossain, "A Semantic Web Approach to VerifyingProduct Line Variant Requirements," in *Fourth International Conference on Advances in Communication, Network and Computing*, 2013, pp. 186–191.

[15] "Protégé Ontology Editor," 2015. [Online]. Available: http://protege.stanford.edu/.

[16] "Pellet," 2015. [Online]. Available: http://pellet.owldl.com/.

[17] X. Dolques, "RCAExplore Relational Concept Analysis and Exploration," 2014. [Online]. Available: http://dolques.free.fr/rcaexplore/.

[18] W3C, "SWRL:A Semantic Web Rule Language Combining OWL and RuleML." [Online]. Available: https://www.w3.org/Submission/SWRL/.

[19] IBM, "An architectural blueprint for autonomic computing," *IBM White Pap.*, vol. 36, no. June, p. 34, 2006.

[20] SWI-Prolog, "SWI-Prolog." [Online]. Available: http://www.swi-prolog.org/.