Multi-View Variability Modeling for Distributed Deployment in Extended Smart Home Architecture

Amal Tahri [1,2], Laurence Duchien[2], Jacques Pulou[1]

[1]Orange Labs Meylan, France
[2]INRIA Lille-Nord Europe, CRISTAL laboratory, University Lille 1, France

Abstract.

1 Introduction

Software Product Lines (SPL) based on variability management are rapidly emerging as an important software development paradigm in complex environment. SPL development approach has proven its efficiency in the last decades for building product lines with higher quality and at lower cost. But along with the gains come risks. Using a product line approach constitutes a new technical strategy that affects the software development and deployment processes. The deployment of a software product line is the mapping of a set of software features of a product on a set of deployment nodes. The challenges of such deployment is managing the variability of the software application product line and the variability of the deployment node product lines.
In a context such as the Smart Home (SH), the raise of new automation services such as home security, comfort and energy efficiency offers a multitude of options and choices to the customers of the SH. These services, delivered as component-based applications [13] can be represented as product lines. However, the SH market, nowadays, is driven by the technological fragmentation: Service providers are scattered over the SH market to offer vertically integrated end-to-end solutions. Each provider offers isolated silo of solution with their own deployment environment such as automation boxes and home devices, e.g., sensor, actuator, camera. To our opinion, actual deployment environment limits the SH expansion by adding new devices and boxes which increase dramatically the SH cost and therefore the SH complexity.
In order to limit this complexity, we promote the Smart Home Extended Architecture (SHEA), an open platform that expands the SH deployment environment to include the Cloud platform and mobile connected devices to host the SH applications. SHEA encompasses different deployment nodes such as the Home Automation Box (HAB), Cloud nodes on the Platform as a Service (PaaS), other connected things and mobile phones. These nodes are the target platform for hosting SH applications. However, SHEA is characterized by the variability of the applications and the variability of the available deployment nodes. Application variability refers to different options and choices of the same application

that delivers the subscribed service by the customer. Deployment node variability refers to different resource offers: from embedded node inside the SH with limited resources to virtual node on the Cloud PaaS with on-demand resource allocation. Disposing of several nodes dispatched between the SH and the Cloud enlarges the deployment configurations for the application.
Therefore, the first challenge is to ($C1$) express and manage the variability of the application and the variability of the available nodes in the SHEA in order to identify all the valid deployment configurations.
Different stakeholders are involved in the SHEA. Service providers offer different services to SH customers. Telecommunication operators offer the Home Automation Box (HAB), the centric deployment node in the SH that connects the different home devices, e.g., sensors, actuators and other connected things to the home network as well as other devices, e.g., mobiles or tablets. Cloud providers offer computational and storage capacities thanks to the PaaS and its on-demand resource allocation. Customers are the target clients of the service providers, telecommunication operators and Cloud providers.
However, each stakeholder has specific concerns which describe multi-views in the SHEA. Therefore, to offer an open environment to different stakeholders, an unified modeling approach is needed to express and understand each stakeholder concern and the related views. Then, the second challenge is ($C2$) to involve all the different stakeholders in a modeling approach that presents and describes the different views related to the SHEA stakeholder concerns.
In this paper, we consider the deployment of a SH application product line on multiple deployment node product lines in the SHEA with specific consideration of the variability and the multi-view properties of the SHEA. In Section 2, we propose a multi-view variability meta-model based on the IEEE P1471 [2] Recommended Practice for Architectural Documentation with respect to product lines in general, and architectural and feature variabilities in particular. Section 3 illustrates the proposed variability documentation applied to the deployment view of the "4+1" view model [6]. In particular, we focus on the deployment view of a software architecture of a SH application product line by showing the architectural representation in feature modeling and how the different views e.g., logical, process and development contribute to the definition of the deployment view. Section 4 shows the effective deployment of this example using an autonomic deployment platform in the SHEA. Related work is described in Section 5. Finally, conclusion and future works are presented in Section 6.

2 Multi-view Variability Meta-Model

In this section, we introduce the Smart Home Extended Architecture Deployment Framework (SHEAD-F) that supports the deployment of an application product line on multiple product lines of deployment nodes in a complex environment. This framework is based on a multi-view variability meta-modeling approach presented in Figure 1, to provide guidance between the development process and the deployment process of a software product line. To avoid rein-

venting the wheel, the multi-view variability meta-model is an extension of the architectural variability meta-model presented in [14]. The novelty of this approach is the description of an architecture of a software product line with specific consideration of the multi-view and the variability properties for the deployment purpose.
The expression of these properties are introduced through three extensions:

1. Product Line Extension (PLE) is shown in the upper left of the Figure 1.
2. Architectural Variability Extension (AVE) covers the explicit representation and specification of variability for the deployment purpose in the architecture of a product line.
3. Feature Variability Extension (FVE) represents how to express variability using extended feature modeling with specific consideration of the deployment purpose.

2.1 IEEE P1471 and Product Line Extension

The IEEE P1471 recommended practice for architectural description, shown in the upper right of the Figure 1, defines the architectural description from different stakeholder concerns, views, and viewpoints.
The P1471 is focused on single products and does not cover the modeling of product lines. Therefore, we propose an extension to cover the variability of a product line through an architectural variability extension and a feature variability extension. Such extensions have been presented in [14] with focus on the design phase of a product line. In this paper, we focus on the missing elements to define the deployment phase.
IEEE P1471 defines the description of an architecture of a `product line` as shown in the PLE in the upper left of the Figure. A software product line is "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [**?**]. A product line has an `architecture` and can be instantiated as `product`. A product line has one or several `stakeholder`, in the IEEE P1471, with a specific `concern`. The stakeholder addresses a specific `viewpoint` which is recorded in a `library viewpoint`. `Architecture Description` has one or several `Architectural view`(s) described in `Architectural View Model`. A detailed description of the IEEE P1471 can be found in [2].

2.2 Feature Variability Extension

Several works on the variability modeling have proposed a meta-model to describe the use of feature models with an abstract syntax [11, 8, 10].
A feature is understood as "a product characteristic that users and customers view as important in describing and distinguishing members of the product line"[14]. In this context, features represent deployment units and thus describe the problem space for the product line deployment phase.

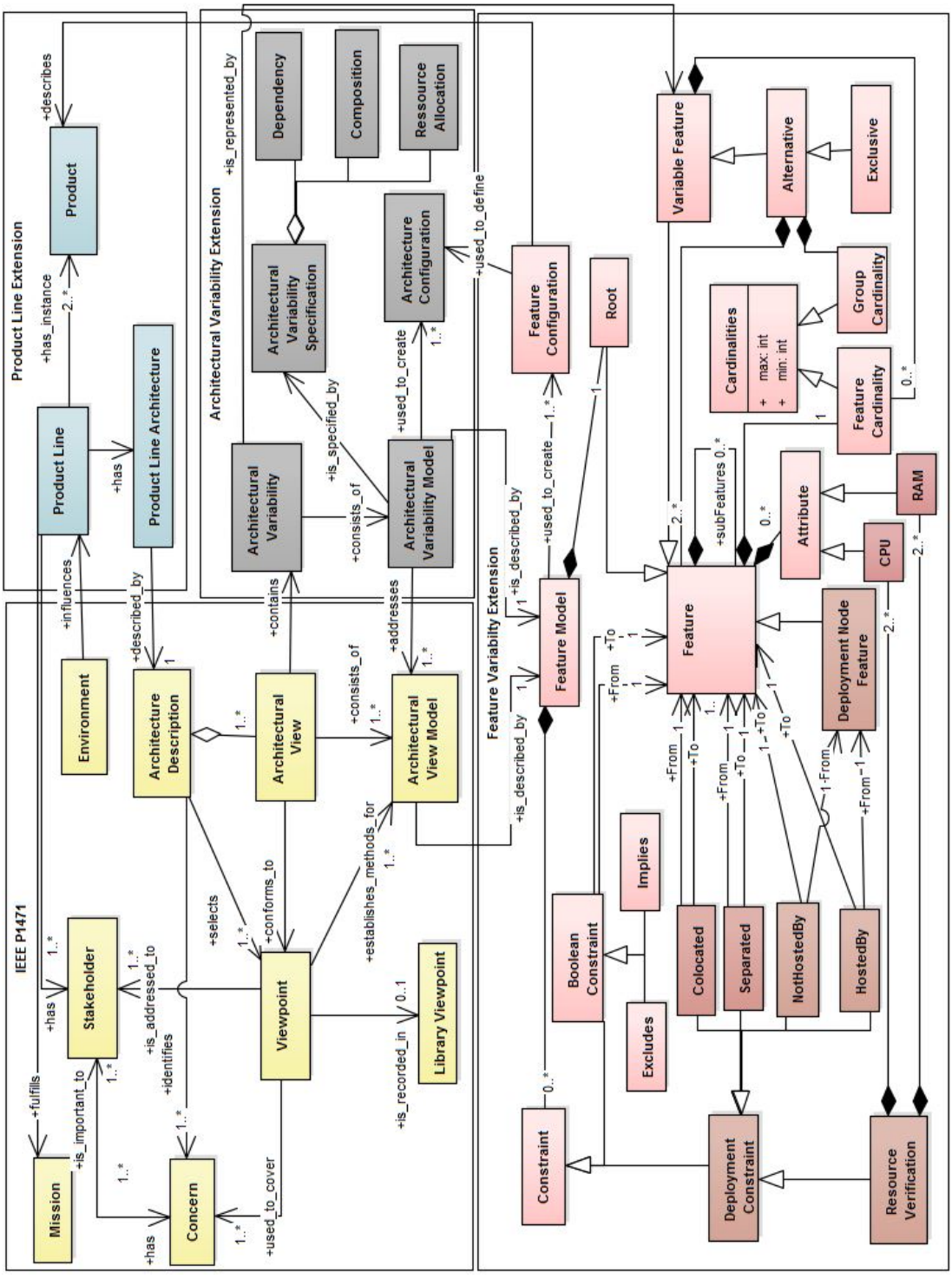

Fig. 1. Multi-View Variability Meta-Model.

Variability meta-model, in Figure 1, refers to `Feature Model` meta-class composed of a `root` and `constraints`. Root is derived from the `Feature` meta-class. Each feature has `subFeatures`, `cardinalities` and `attributes`. Cardinalities indicate how many instances of the feature can be present in a configuration. A feature can be either common or *variable*. In the first case, the feature is included in all *feature configurations* to describe a *product* in a *product line* (Product line Extension in Figure 1). In the latter case, the feature is implemented in a single feature or a set of features (at least 2), but not by all. These members can be `Alternative`, which can be an `Exclusive`. Alternative allows the selection of one or several sub-features in a configuration. Exclusive indicates that only one sub-feature can be selected in a configuration.
Dependencies among features are represented using *constraint* . "*Boolean constraint* are *implies* and *excludes*: Implies signifies that when a *From* feature is selected, the *To* feature must be present in the same configuration. Excludes means that if *From* feature is selected, the *To* feature must not be present in the same configuration.
We have introduced the `Deployment Node Feature` as a new feature category to represent the deployment node. `Deployment Constraint` is a new constraint category that expresses fundamental deployment constraints such as `Colocated Seperated` between two features, and `HostedBy` or `NotHostedBy` between a feature and a deployment node feature.

2.3 Architectural Variability Extension

Architectural variability can be summarized in an *architectural variability model* that can be used to create *architecture configuration(s)*. An architecture configuration is defined as a *feature configuration* in the FVE that describes a product in the PLE. To define an architecture variability model, the *architectural variability specification* has to be described using the *feature model* from the FVE. This specification is expressed as *dependency*, *composition* and *resource allocation*. Dependency is related to *constraint* in the FVE that describes the relationship among feature. Composition defines the hierarchy of *feature model* in the FVE. This hierarchy is described as *feature*, *subFeatures* and *variable feature*. Resource allocation is expressed as *attributes* and the resource allocation constraint in the FVE.
For a product line, *architectural variability* is present in each architectural view and therefore each ARCHITECTURAL VARIABILITY MODEL addresses an architectural view model. Architectural variability is represented by a *variable feature* in the feature variability extension.

3 Modeling the Deployment View Variability

In this section, we show examples from applying the multi-view variability modeling concept introduced in the previous section to a product line of SH application to be deployment on several deployment node product lines in the SHEA.

We focus on the deployment view in with consideration of the (i) variability of the application and the variability of the deployment nodes and (ii) multi-views related to different stakeholder concerns.

3.1 Motivation Example

Security service provider offers a "Motion-Sensing Detection" application for home security. This application records video flows from a security camera, and takes snapshots of the detected changes in the video flow. These snapshots can be analyzed in order to identify and match faces with a predefined data base of friends and families of the home owners, i.e., customers. Live streaming from the camera is offered with the basic product. Face recognition is only available with the premium product. Premium product can have different variants with different performances due to the used recognition algorithm. The security service provider aims to launch new products quickly by reducing the time-to-market. Therefore, the provider adapts a Software Product Line (SPL) development approach.
Telecommunication operators and Cloud providers deliver the product lines of the deployment nodes, respectively, the Home Automation product line and the Cloud PaaS product line. This application is displayed to the SHEA customers through the SHEA store. This store allows the customers to purchase a new service, select the preferred product, i.e., basic, premium, from the SH store functionalities and install remotely the application on the existing nodes in their homes. The variability between the different products are the selection of the following functionalities: "*Motion Detection*", "*Face Recognition*", "*Live Streaming*" and "*Storage*". *Motion Detection* captures and streams video flows from a surveillance camera. It detects changes of movements in the video flow. Then, it takes snapshots of these changes and displays them to customer screen(s) as *Live Streaming*. The *Storage* of the video flow is done automatically. *Face Recognition* extracts and identifies face images from a video flow of a surveillance camera. The images are captured and submitted for face recognition where images are matched with face images in the data base of authorized persons. Finally, a notification warns the customer when a movement is detected and/or a person is identified.

3.2 Challenges

3.3 Introducing Variability to the Deployment View

Assumptions

Modeling Approach The modeling approach using the "4+1" view model is an interesting approach that involves several stakeholders to produce only one application. However, the "4+1" view model is not adapted to express the application variability related to each view. A product family of SH application deployed on product families of deployment nodes in the SHEA are considered.

Roles and inputs Roles : architects, integrateurs, et deployeurs qui definit les contraintes de deployement. le developpers n'a pas vraiment d'importance.
Software architects describe the Functional Requirements (FR) of the application: What the SH application should provide in terms of services to its customers. Developers implement the application. Integrators verify the dynamic behavior and the Non-Functional Requirements (NFR) of the application. Deployers deploy the application on the customer deployment nodes.

Resource Allocation The postulate of this paper is that "deployment units express their resource requirements, e.g., CPU and RAM to meet specific QoS. If the deployment nodes dispose and can satisfy the deployment unit requirements, then, deployment nodes automatically allocate and guarantee the required resources to the deployment units. If deployment nodes cannot guarantee the deployment unit requirements, they refuse the resource allocation". The node resources can satisfy the deployment unit demands regarding of the resource allocation model. For example, a process allocation model is based on different threads and the Round Robing (RR) dispatching algorithm. The CPU scheduler goes through a waiting queue and allocates the CPU to thread i for a time interval of ki quanta. Suppose now that the RR period is n quanta and that CPU gets a full computation power of X MIPS. If a process claims C MIPS of CPU with respect to NFR, so the CPU scheduler should allocate $ki = C * n/X$ quanta.
This postulate explains how component from development view respect the NFR that comes along with the resource allocation, thus verifying the postulate.

Deployment View Variability

4 Preliminary Validation

4.1 Consistency checking

4.2 Supporting the Configuration Process

5 Related Work

6 Conclusion

References

1. D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.
2. G. Booch, I. Jacobson, and J. Rumbaugh. *The unified software development process*, volume 1. Addison-wesley Reading, 1999.
3. K. Czarnecki, C. Hwan, P. Kim, and K. Kalleberg. Feature models are views on ontologies. In *Software Product Line Conference, 2006 10th International*, pages 41–51. IEEE, 2006.

4. N. Jussien, G. Rochart, and X. Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.
5. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.
6. P. B. Kruchten. The 4+ 1 view model of architecture. *Software, IEEE*, 12(6):42–50, 1995.
7. J. Muskens, M. Chaudron, and C. Lange. Investigations in applying metrics to multi-view architecture models. In *Euromicro Conference, 2004. Proceedings. 30th*, pages 372–379. IEEE, 2004.
8. C. Parra. *Towards dynamic software product lines: Unifying design and runtime adaptations*. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2011.
9. C. Pettey and H. Stevens. Gartner highlights key predictions for it organizations and users in 2010 and beyond, 2010.
10. K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering*, volume 10. Springer, 2005.
11. C. Quinton. *Cloud Environment Selection and Configuration: A Software Product Lines-Based Approach*. PhD thesis, Université Lille 1, 2014.
12. C. Quinton, D. Romero, and L. Duchien. Cardinality-based feature models with constraints: a pragmatic approach. In *Proceedings of the 17th International Software Product Line Conference*, pages 162–166. ACM, 2013.
13. C. Szyperski. *Component Software: Beyond Object-oriented Programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2002.
14. S. Thiel and A. Hein. Systematic integration of variability into product line architecture design. In *Software Product Lines*, pages 130–153. Springer, 2002.