A DYNAMIC SOFTWARE PRODUCT LINE FOR REMOTE MONITORING OF COMPUTER SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜLŞAH ERDİL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2019

Approval of the thesis:

A DYNAMIC SOFTWARE PRODUCT LINE FOR REMOTE MONITORING OF COMPUTER SYSTEMS

submitted by **GÜLŞAH ERDİL** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar

Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün

Head of Department, **Computer Engineering** _____

Prof. Dr. Halit Oğuztüzün

Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Assist. Prof. Dr. Ebru Aydın Göl

Computer Engineering, Middle East Technical University _____

Prof. Dr. Halit Oğuztüzün

Computer Engineering, METU _____

Assist. Prof. Dr. Ayça Tarhan

Computer Engineering, Hacettepe University _____

Date: 20.02.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Gülşah Erdil

Signature:

ABSTRACT

A DYNAMIC SOFTWARE PRODUCT LINE FOR REMOTE MONITORING OF COMPUTER SYSTEMS

Erdil, Gülşah
Master of Science, Computer Engineering
Supervisor: Prof. Dr. Halit Oğuztüzün

February 2019, 76 pages

Remote Monitoring and Management systems are Information Technology (IT) software tools to organize and manage distributed client workstations from a central point. They are used by many large-scale technology companies that are willing to minimize their labor cost needed for running the IT infrastructure, collect and measure the data of clients within the organization, administrate them from a single point, in a reliable and secure way. Dynamic user profile deployment, dynamic reconfiguration of remote monitors in response to changes in clients' workstations and creating on-the-fly notifications according to monitor results are the main features of remote monitoring and management systems. These features can be fully implemented by Dynamic Software Product Line (DSPL) Engineering. DSPL for Remote Monitoring of Computer Systems aims to provide IT Service Providers with a dynamically reconfigurable, reusable and easy to define monitoring and measurement mechanism. The presented study offers monitoring mechanisms and it also provides an infrastructure for further management applications. It applies DSPL engineering concepts which are well defined in academic studies but not extensively implemented in business realm. It enables IT management systems with reusable and autonomously monitoring software assets. This study also provides a remote monitoring application

as a case study, constructed by using the aforementioned DSPL with the help of the reusable software components.

Keywords: Remote Monitoring and Management, Variability Management, Dynamic Software Product Line, Reconfigurability

ÖZ

BİLGİSAYAR SİSTEMLERİNİN UZAKTAN İZLENMESİ İÇİN DİNAMİK YAZILIM ÜRÜN HATTI

Erdil, Gülşah
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Danışmanı: Prof. Dr. Halit Oğuztüzün

Şubat 2019, 76 sayfa

Uzaktan izleme ve kontrol sistemleri, dağıtık iş istasyonlarını merkezi bir noktadan organize etmek ve yönetmek için kullanılan Bilgi Teknolojileri (BT) yazılımlarıdır. Büyük ölçekli birçok şirket, bilgi teknolojileri altyapıları için harcanan iş gücünü azaltmak, bu işlemleri tek bir noktadan sağlıklı ve güvenli bir şekilde yönetmek için bu yazılımları kullanmaktadır. Dinamik kullanıcı profili tanımlanması, iş istasyonlarının değişen durumlarına bağlı olarak monitörleme mekanizmalarının dinamik olarak yeniden yapılandırılabilmesi, monitör sonuçlarına göre anında bildirim oluşturulabilmesi, uzaktan izleme ve kontrol sistemlerinin ana özellikleridir. Bu özellikler Dinamik Yazılım Ürün Hattı Mühendisliği (DYÜH) metodolojileri ile tam olarak gerçeklenebilmektedir. Bilgisayar Sistemlerinin Uzaktan Kontrolü için Dinamik Yazılım Ürün Hattı, BT servis sağlayıcılarına dinamik olarak yeniden yapılandırabilen, tekrar kullanılabilir, kolay monitörleme ve yönetme mekanizmaları sunmayı amaçlamaktadır. Bu çalışma esas olarak monitörleme mekanizmalarını sunmaktadır ve dolaylı olarak yönetim mekanizmalarına altyapı sağlamaktadır. Sunulan çalışma, akademik araştırmalarda iyi tanımlanmış, ancak iş dünyasında yaygın olarak uygulanmayan DYÜH kavramlarını hayata geçirir ve BT yönetim sistemleri için tekrar kullanılabilir ve otonom monitörleme yazılım bileşenleri sunar.

Ayrıca bu ürün hattı kullanılarak hazırlanmış bir uzaktan yönetim uygulaması da vaka analizi olarak sunulmaktadır.

Anahtar Kelimeler: Uzaktan İzleme ve Kontrol, Değişkenlik Yönetimi Dinamik Yazılım Ürün Hattı, Yeniden Yapılandırma

To My Beloved Grandmother and Family

ACKNOWLEDGEMENTS

Initially, I would like to thank my supervisor Halit Oğuztüzün, who is always patient, understanding but more importantly a guiding spirit for me. He always led to me to right ways, encouraged me to do the better.

Besides my supervisor, I would like to thank all the colleagues from Nurd Yazılım. They also contributed to my study during the tests and demo preparation and spent time for my study.

I would like to thank Uğur Çakır for the financial and moral support he provided to me for my academic studies. He eased the process to continue my studies as parallel to work life.

Thanks to Sina Entekhabi, for his worthless helps and contribution to my study. I am grateful to him for allowing me to use the algorithm he developed in my thesis.

My husband, Onur Can Erdil is one of the biggest supporters on all stages of my life, and he was again the biggest supporter during the hard days of the study. Special thanks to him for his help and love.

The architects of all the achievements I have reached today is my dear mother, father and sister. I am grateful that I have them and their unrequited love. I always owe them for their unconditional support and all the facilities they have provided to me.

This work has been supported by TÜBİTAK-ARDEB-1001 program under project 215E188.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

C1	Comodo ONE
CO	Control Objective
COBIT	Control Objectives for Information and Related Technology
DAS	Dynamically Adaptive System
DCO	Detailed Control Objective
DSPL	Dynamic Software Product Line
DSPLE	Dynamic Software Product Line Engineering
FM	Feature Model
FODA	Feature Oriented Domain Analysis
HIPS	Host Intrusion Prevention System
IT	Information Technology
ITIL	Information Technology Infrastructure Library
ITSM	Information Technology Service Management
MoRE	Model-Based Reconfiguration Engine
MSP	Managed Service Provider
PLE	Product Line Engineering
RM	Remote Monitoring
RMM	Remote Monitoring and Management
SLA	Service Level Agreement
SPL	Software Product Line
SPLE	Software Product Line Engineering
UAR	Unknown Application Running

CHAPTER 1

INTRODUCTION

Fast evolving world and dynamic adaptation needs to catch up continuously changing technology trends has brought large scale, multi employee companies along with. To ensure the sustainability of the work flow, to increase productivity by keeping employee's workstations healthy and consistent, and to keep servers, networks and software applications of the company always up and running, companies hire Information Technology (IT) staff and managers. To reduce the cost to employ IT management staff within the organization, but more importantly, to diminish the human errors, to automate the management of computer systems and to administer them from a single center, Remote Monitoring and Management (RMM) software tools are being used by organizations. These tools are not only used directly inside the IT departments of technology companies, but also used by Managed Service Providers (MSP). Several large companies do not even have an IT department, but they outsource their IT management operations to professional MSPs. Companies that are willing to enhance the quality of their services and reduce the risks for their product development, they budget for professional Managed Service Providers. MSP's are responsible to fulfill all technological needs in IT sense such as managing endpoints or servers' network infrastructure, clients' portals, their troubleshooting and maintenance. To fulfill these aims, MSPs need to distribute their services to several remote customers across a wide network, generally via internet (Kumbakara, 2008). At this point, RMM software are perfectly matched for this requirement and they are started to be used by many MSP companies.

RMM software generally has two components, one of which is an admin portal, used by IT personnel and the other is RMM agent deployed to managed client computers. With the help of the agents, RMM system identifies newly joined clients and deploys

some default and predefined set of monitoring mechanisms and instructions. There is also need for RMM systems to take actions after deployment and show dynamic behaviors during runtime. In this regard, the application of Dynamic Software Product Line (DSPL) Engineering methodology seems well suited for RMM systems. Although there is extensive academic research on DSPL concepts and techniques, the application areas and best practices in business are still preliminary. How DSPL mechanisms are put into practice in RMM product families and what are the effectiveness, benefits and gains in DSPL approach applied to RMM systems are the driving questions for this thesis.

A Remote Monitoring and Management System is used to manage clients' workstations remotely with the help of automatic, reliable and maintainable management monitors. The features of RMM are varying according to the custom needs. Nevertheless, in all of them, RMM can detect the new clients joining the network and then it configures and manages them automatically by the rules prescribed by IT service providers. For example, after new client identification is completed, a default profile defined by IT manager or a specific one to this client is deployed to the end user machine and controls it remotely. In a security concerned profile, the manager can define such a rule that Malware Detection monitor starts to watch the computer and whenever it detects a virus or spy, it removes the malware automatically, or creates a notification to the user or Management Service Providers. As another rule example, whenever a CPU Monitor detects that CPU usage is exceeding the 70 percent, then Event Log Monitor also is set up. Considering these features, remote monitoring and management software is very suitable to model as a Dynamic Software Product Line. With this motivation, DSPL of Remote Monitoring Systems emerged, which facilitates the construction and configuration of remote monitors by applying the dynamic reconfiguration and asset reuse approaches guided by a variability model.

This study contributes to the software engineering field by exploring the applicability of DSPL methodology in the Remote Monitoring and Management domain. Although

computer systems are addressed specifically on this study, it can also be a guide for monitoring systems in other application fields. The study focuses the Remote Monitoring (RM) area mainly and management part is out of concern for this thesis. Management components and related software assets will be added to the framework as a future work. After this point, only remote monitoring concepts will be mentioned, and related software components will be represented.

The outline of this work is as follows: Initially the introduction part including the motivation behind the study will be given as Chapter 1. Chapter 2 provides the background of RMM and DSPL concepts. Then commonality and variation points analysis of RMM systems is given under the Chapter 3 Feature Model of Remote Monitoring Systems, which is an important phase for dynamic software product line development. Chapter 4 specifies detailed design for DSPL in terms of architecture, self-awareness, dynamic binding mechanism and reusable software assets. In Chapter 5, as a case study COMODO ONE ITSM Product generated with the help of our framework is explained. As a final section, at Chapter 6 conclusion and future research directions are presented.

CHAPTER 2

BACKGROUND

2.1. Summary

This chapter introduces background information about the study. It enlightens the basics of two concepts that are subject to this thesis: Information Technology Service Management and Dynamic Software Product Lines. In the first part, ITSM essentials, standards, approaches and RMM-as-IT-service are explained. Under DSPL subsection, SPL as building block, Variability Management techniques and Dynamic Reconfiguration need in SPL engineering are discussed.

2.2. Information Technology Service Management

Today's companies are dealing with considerable amount of data to carry out their production, marketing or maintenance processes. To sustain their product lines and increase the quality of their services, they need to build an infrastructure to create, store, modify and exchange these data in a secure and reliable way. This need has become crucial for firms because only a healthy business development environment allows for the development of healthy products. Therefore, companies attach great importance to the management of information technologies and make serious investments on IT management. As computer-based technology firms are spread worldwide, the diversity of information and the power of data increased, their management also became diverse and challenging. Almost every company either have created a separate and specialized IT management department, or for especially large-scale and cross-country firms have been outsourcing this need to professional IT management companies or Managed Service Providers (Sena & Obispo, 2006). Keeping the IT management infrastructure stable and consistent is vital for firms in almost every sector, because it directly affects the business development environment

and the resulting quality of product or services. Companies know that any failure or outage on IT infrastructure leads to outages on data flowing inside networks, failure on internal business processes and resulting product in return. Any unavailability of servers of a technology company, hardware or software problems on employees' machines or unpredictable interruptions on network give rise to failure on business performance and impair the product (Hinz & Gewald, 2006).

2.2.1. Information Technology Service Management Approaches and Standards

2.2.1.1. ITIL

To provide a standard development and operating environment, handle complex and diverse business activities, IT Service Management (ITSM) approaches and set of standard IT services are shaped since information technology has taken over the world. Information technology Infrastructure Library (ITIL) is one of the *de facto* standards and it has been continuously revised as IT management challenges increase. ITIL is IT service management framework and it defines IT service concepts, provides best practices for planning of these services and delivery of them to companies. It also highlights the methodologies about the service support after its delivery is completed. ITIL specifies services as something that a customer, who can be the owner of a technology company, needs and utilizes to maintain their business. For example; an IT service can be a server, database or storage administration, network management or regulation of user workstations (Arraj, 2010).

ITIL defines the IT service standards in a 4-step life cycle. First step is *Service Strategy* in which general approach is set by understanding the customer IT service needs, what are the customer specific requirements, required IT capabilities and infrastructure to meet these requirements. This step includes high level design of development and implementation phases and covers other steps of framework. *Service Design* step ensures that before delivery of services, they should be designed to fit exactly the customer needs in a cost-effective way. Necessary technology and tools should be determined on this step. *Service Operation* as a third step aims to deliver designed

services to customer, check the health status of these services and provide a nonstop IT infrastructure. *Continual Service Improvement* is last but most important phase and on this step the effectiveness of provided services are measured, renewing or improvement of IT services are done, and new approaches are applied to existing services (Arraj, 2010).

2.2.1.2. COBIT

The Control Objectives for Information and Related Technology (COBIT) is another framework created by Information Systems Audit and Control Association (ISACA), which defines a set of control mechanisms over IT services and used by many organizations in the world. The framework's main aim is to align IT processes within the company with the business goals via provided controlling mechanisms.

COBIT argues that proper IT government and effective use of IT services are directly linked with the achievement of business goals of technology companies. Application of COBIT makes a significant contribution on business achievements within the firms, via the Control Objectives (CO) defined inside framework (Ridley, Young, & Carroll, 2004). These control objectives are generated for alignment of business objectives and IT processes, and these processes are grouped into four different domains: Planning & Organization, Acquisition & Implementation, Delivery & Support, Monitoring and Evaluating (Sahibudin, Sharifi, & Ayat, 2008).

The framework defines 34 different control objectives, and several detailed control objectives (DCOs) for each CO to fulfill IT governance and management. It is claimed that, appropriate application of these objectives results in successful IT governance and business achievements in return (Von Solms, 2005).

2.2.1.3. ISO/IEC 27000-series

The ISO/IEC 27000 series is set of information security standards published by International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). These standards provide a methodology and

systematic approach to keep information assets secure and provides security specific control objectives to companies implementing these standards (Sahibudin, Sharifi, & Ayat, 2008). These standards provide "how to" specifications in addition to "what to" information about the IT management including capabilities, infrastructures and methodologies, especially on security domain.

2.2.2. Remote Monitoring as an IT Service

Information technology services can be provided by internal IT departments as well as they can be outsourced to third party organizations such as Managed Service Providers (MSP). In both approaches, professional and successful IT service management is used as a tool for business accomplishments. To achieve these accomplishments, required IT services must be correctly identified by organizations and MSPs should provide perfectly matching IT services to customers. To fulfill these two crucial needs, Service Level Agreement (SLA) protocols are created between Service Providers and companies.

IT services delivered by service providers are grouped into 7 different domains (Kumbakara, 2008):

- Help Desk: Centralized services to fulfill IT related end-user requests and problems belong to this domain, services like incident solving, change request handling can be provided in email, phone call or face to face methods.

- Asset Management: Services for physical inventory, budget or software licensing inside the enterprise fall into this domain.

- Procurement: Hardware or software purchasing, pricing, ordering and maintenance during hardware/software's lifecycle related services are procurement services.

- Service Management: Health check of running IT assets, status change or problem monitoring, reporting and diagnosing services are included.

- Security: Intrusion detection over local network areas, virus/spy protection, detection of firewall activities, user account management, password management are all referred as security concerned services.

- Network Monitoring & Administration: Under this category, monitoring of network traffic, health check of devices over a network area, network error detection and fix, LAN & WLAN management, and performance concerned services are listed.

- Hardware & Software Support: Fulfillment of hardware and software service needs, install, update or uninstall operations, maintenance of these assets are hardware & software support services. These services target keeping the end-users' work environments healthy.

Remote Monitoring and Management is a software tool used by IT Management service providers by applying some of the services on mentioned domains. It manages and organizes client systems especially on Security, Network Monitoring, and Service Management domains. RMM system enables MSPs to setup monitors over client computers/systems to gather information and data pertaining to how the system is performing and to supervise the security, performance and network related issues. With the help of the monitoring mechanism, the MSP or the technical support team can execute management tasks autonomously (SearchITChannel, 2015). Remote Monitors are, in general, shaped with following concerns: Energy, Security, Performance and Resource Utilization. Under these categories; TCP, Network Bandwidth, CPU, RAM, Folder Size, File Size, Intrusion Detection and many other monitors are set to clients' computers and their workstations are managed by service providers remotely. An autonomous and self-adaptive remote monitoring and management software is preferred by many companies because it helps to reduce labor cost and replace the error prone manual IT service management operations. Instead of wasting hours for employed IT support teams, RMM software with a single IT manager can provide several IT services for different service domains. It also enables

IT Manager / MSPs to organize the client machines automatically by minimizing human factor and solves the issues without interrupting the client.

With the help of RMM software, IT service providers can apply proven standards and approaches like ITIL or ISO/IEC 27000-series. RMM provides an admin portal for MSP, to prepare the set of rules and monitoring mechanisms for end users' workstations with "always-eyes-on", 24 hours per day watching paradigm. This portal is named as ITSM Portal and all devices within the organization together with their information are listed visually to MSP. Service providers can prepare grouped monitors specific to different domains, like Security or Network Administration, with the help of ITSM portal, and then assign them to a group of end-users having similar profiles. To deliver these monitoring needs, an RMM Agent is installed to client computers within the network of the company. This agent works silently without interrupting the user and monitors the user's work environment according to the rules defined by MSP.

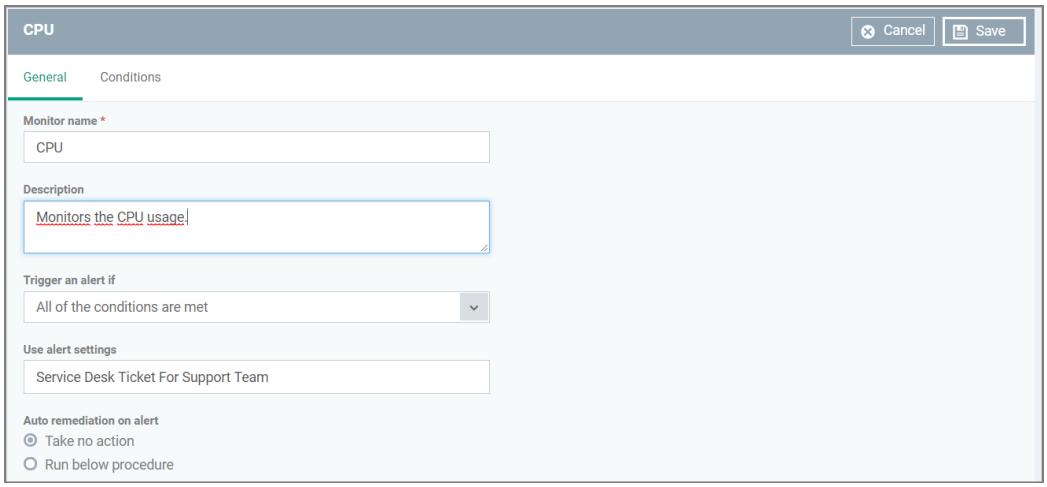

Figure 1 Example CPU Monitor Creation (Courtesy NURD)

An example CPU monitor creation page from NURD's COMODO ONE ITSM Portal can be seen on Figure 1. In addition, corresponding rule definition for this CPU monitor can be observed on Figure 2. During monitor creation, MSP is also able to attach an alerting mechanism which is triggered when the requested condition is met (See Figure 3). For example, on the devices applied CPU monitor, a service

desk ticket is created for Support team when the CPU usage is more than or equal to the given threshold, i.e. 100% for 5 minutes. This ticket also contains the current threshold value, device data and performance metrics information of this device. Such predefined monitors on ITSM Portal can be deployed to many devices on network automatically and then the support team can examine the alerts created by these monitors and take an action to solve the problem on the end user machine. Therefore, this is an autonomous, cost effective, reliable and scalable IT service delivery method.

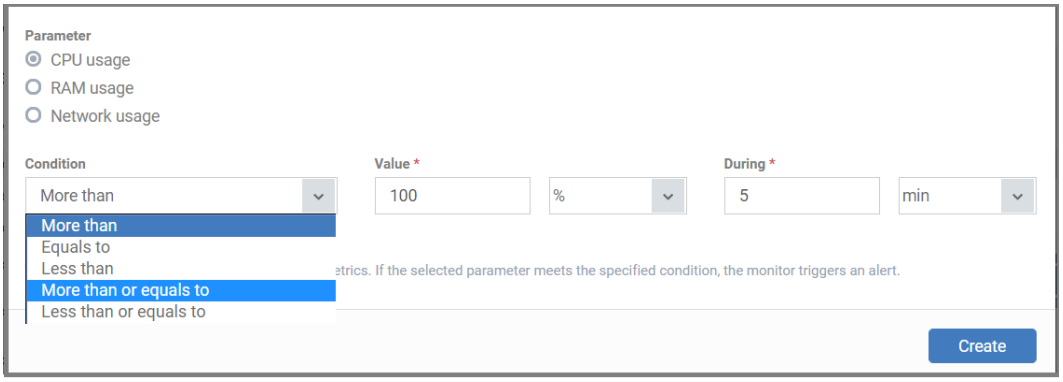

Figure 2 Example CPU Monitor Rule Definition (Courtesy NURD)

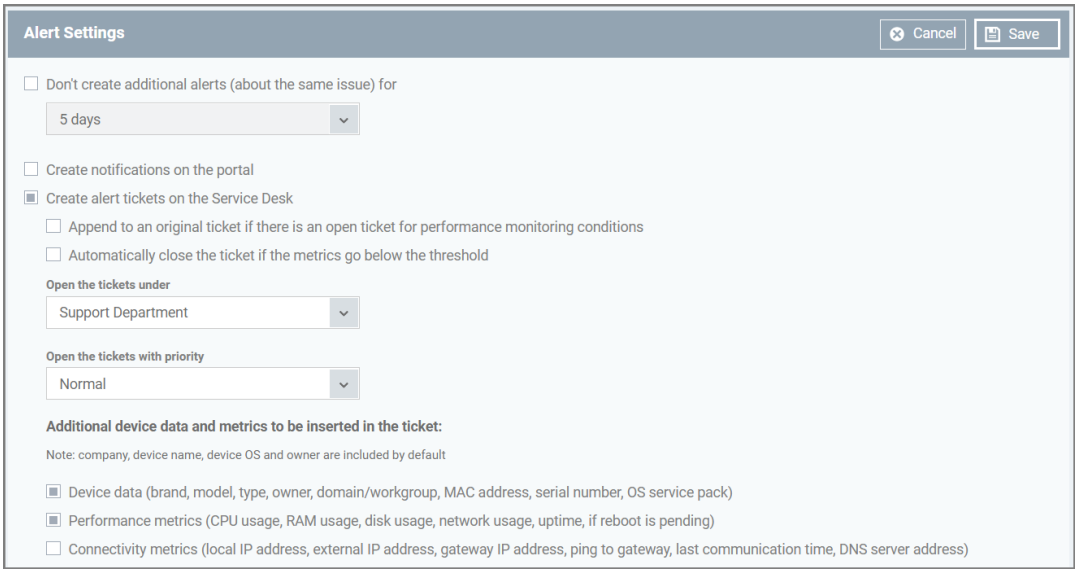

Figure 3 Example Alert Creation for Service Desk Support Team (Courtesy NURD)

2.3. Dynamic Software Product Lines

2.3.1. Software Product Lines

It has started to be more challenging for the companies that their customers request for changes and modifications continuously for the software intensive systems they are acquiring. To fulfill these requests and cope with the challenge of customer specific adaptation, firms can produce short term solutions like “working-hard” or increasing work power, but in long term these lead to loss of time and money. Such solutions are also reasons behind the failure of companies among their competitors (Capilla, Bosch, & Kyo-Chul, 2013). As a long-term solution, Product Line Engineering (PLE) approach which has been already followed for mechanical or physical systems for years, has been started to be used for software applications.

PLE techniques applied in several different projects, aims to create a common architecture and a set of reusable assets. The use of these artifacts on different products from same family reduces time, cost and work load in return (Böckle, et al., 2002, August). Software Product Line Engineering (SPLE) has also same objectives; building a software product family which share the similar software architectures, with similar attributes and core software assets. Besides, the variation points among products and derivation use cases should be identified (Bagheri, Di Noia, Ragone, & Gasevic, 2010). Such a product family built on top of the SPLE basics can respond instantly to customer specific modification and adaptation requests by tailoring the core assets according to specific requirements. These techniques empower firms to produce more qualified products with low cost and man power and improves mass production of software intensive systems.

SPLE is a two-part engineering process: Domain Engineering and Application Engineering. Domain Engineering aims to establish a platform composed of reusable software assets to produce software applications. It comprises all phases of software development such as requirement analysis, architectural design, component

realization and test. It produces reusable assets for all these phases by applying commonality and variability (C&V) analysis over product families.

> *"Domain engineering is the process of software product line engineering in which the commonality and variability of the product line are defined and realized."* (Böckle, Pohl, & van der Linden, 2005)

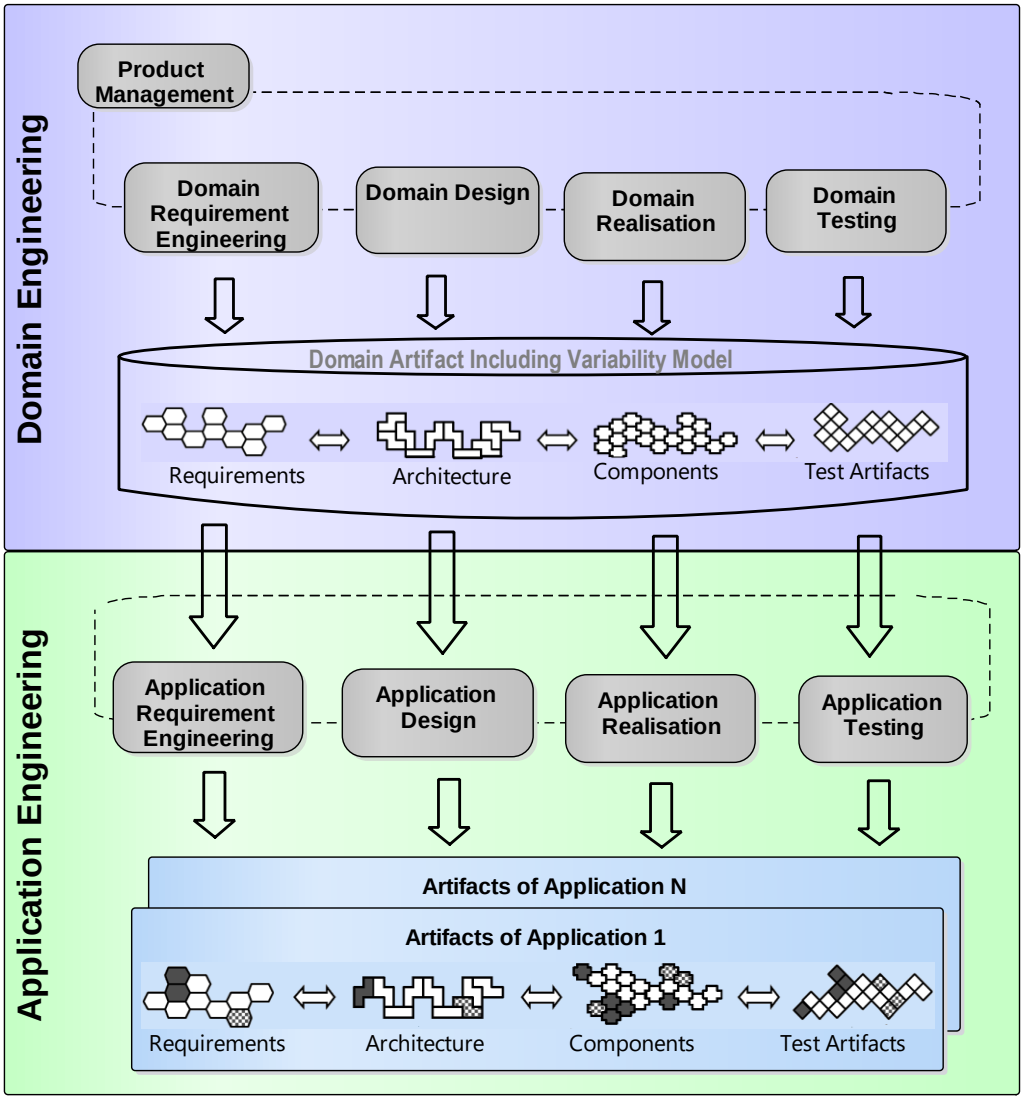

Figure 4 SPL Framework adapted from (Böckle, Pohl, & van der Linden, 2005)

Domain Engineering consists of 5 stages: Product Management, Requirement Analysis of Domain, Design of Domain, Realization of Domain and Test. All these

phases aim to create an infrastructure for a product family, without taking customer or product specific requirements into consideration.

Application Engineering is the phase of binding application and customer specific requirements to variability points defined on the established platform at domain engineering phase. It enables the construction of specialized end products with the help of reusable software components of product line.

> *"Application engineering is the process of software product line engineering in which applications of the product line are built by reusing domain artifacts and exploiting the product line variability."* (Böckle, Pohl, & van der Linden, 2005)

In application engineering, the same 4 steps as in Domain engineering, other than product management are also executed (See Figure 4). But at application engineering phase, set of requirements specific to the product is determined, specific architectural design decisions for this product are applied, the set of reusable components defined earlier is used for product construction and explicit test cases for new product is chosen.

This two-part engineering approach separates SPLE concerns into two parts also: 1) Establishing efficient and powerful product lines, 2) Fulfilling customers' specific needs quickly and manufacturing qualified and low-cost software systems via these product lines. This approach does not need to be executed as waterfall model, instead it is better to apply spiral, V model or agile methodologies as the companies adopt during their software development lifecycle (Metzger & Pohl, 2007).

2.3.2. Variability Management

Variability Managing (VM) is the basis for many Software Product Line Engineering methodologies. VM targets to identify the variable and common aspects among the products that show similar characteristics. Commonality and Variability Analysis is the most used method for variability management. C&V analysis provides software

engineers with a systematic way on the construction of product families. In this systematic approach, commonality implies the attributes that have same values among a set of objects, while variability stands for the attributes that take different values among them (Coplien, Hoffman, & Weiss, 1998).

Determination of variable and common points on the construction of different products is accomplished firstly on the requirement analysis phase of the domain engineering. Requirement analysis provides a great deal of input to the identification of common and variable features for the software product families. Product features are created based on these requirements (Böckle, Pohl, & van der Linden, 2005). After requirements analysis phase, architectural design, component realization and test steps also contribute to feature set construction. At every phase, variation points are refined, and new points are added to the set. In the same steps of application engineering phase, already defined variable and common attributes are bound to real applications. Variability binding means to choose variation points and their corresponding values to generate customized products.

To summarize, for a customized product X belonging to a product family, specific requirements are chosen. Then, design decision that must be applied to X defined on general architecture (on domain engineering phase) is determined. Set of common software assets needed to build X are listed. Then reusable test cases to ensure the quality of the product are selected. By following these steps on the construction of software intensive systems, companies create product lines maintaining low cost product manufacture in short time.

There are various methodologies to identify the common and variation points of product family, but three of them are widely used by technology producing companies: Decision Models, Orthogonal Variability Model and Feature Models.

2.3.2.1. Decision Models

Construction of variation and commonality points for similar software systems is shaped with the design decisions of the stakeholders effective on product creation. This approach is named decision-oriented variability management. In this approach, forming a customized product from a product line is accomplished by taking decisions for every variation point (Dhungana, Rabiser, & Grünbacher, 2007). These variation points arise from differences in the different stakeholders' view of the product. Product is shaped with every decision taken during the both domain and application engineering phases, therefore customization is maintained by stakeholders' decisions at the end of the day.

Decision model is generated during the requirement analysis and architectural design phases of domain engineering and decisions with their corresponding values are resolved by application engineers during end product construction. Decision model shows the extent of the variability for a product line comprehensively (Schmid & John, 2004).

According to the decision model definition in (Schmid & John, 2004), a decision point is the sole determiner for a variant but same decision point can be a determiner for distinct variation points. Each decision point has a name, description, and range of values. Moreover, decision points can be relevant with each other, can require or eliminate others. Therefore, decision models describe relational dependencies. It also shows constraints like “Decision X is made only if Decision Y is bigger than some threshold”. Binding time constrains can also be specified on these models.

2.3.2.2. Orthogonal Variability Model (OVM)

Orthogonal Variability Model provides a cross-sectional relationship between the variability points produced on different artifacts of software development life cycle, such as feature model, use case model or component models (Lauenroth & Pohl, 2005). Orthogonal VM has two base constructs, one of which is Variant, and the other

is Variation Points as it is illustrated on Figure 5. Variation Point is an abstract definition for specialized Internal and External Variation Points. According to the orthogonal VM approach, a variation point can either be internal or external. Internal variation point is visible only for developers while external variation point is visible for both developers and customers. Variation point and variant has dependency relationship which means each variation point must be in correlation with at least one variant, and vice versa. Variability dependency can be optional or mandatory. Optional variability dependency implies that a variant can exist on a specialized product of a product line but does not need to exist. Mandatory relationship means that a variant must exist for an application if the corresponding variation point exists for a customized product. This explanation does not imply that a variant is a part of the products of a family.

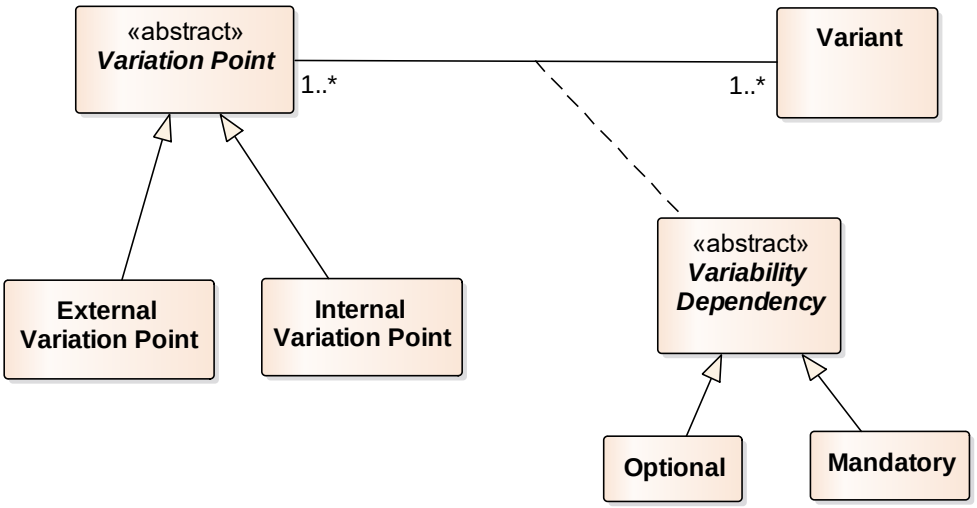

Figure 5 Orthogonal Variability Model Components and Relationships (Lauenroth & Pohl, 2005)

2.3.2.3. Feature Models

Feature Modeling approach had been initiated by Kyo C. Kang and his colleagues as a part of Feature Oriented Domain Analysis (FODA) Feasibility Study (Kang, Cohen, Hess, Novak, & Peterson, 1990). According to the study, to ensure that large and

complex software intensive systems are understood by all stakeholders, firstly their capabilities and features should be clarified. FODA holds that effective reuse of software components can be accomplished firstly by exploring the capabilities and feature set of product family during domain analysis.

Feature and capability definitions are abstractions of product components which can be easily comprehended by both customers and developers. This is the reason behind the extensive use of feature modeling for the commonality and variability analysis of SPLE applications (Kang, Lee, & Donohoe, 2002).

After introduction of feature modeling by FODA, several extensions of this methodology was published in following years: FORM (Kang, et al., 1998), FeatuRSEB (Griss, Favaro, & d'Alessandro, 1998), and Generative Programming (GP) Feature Models (Czarnecki & Eisenecker, 2000) and many more. They can be grouped under three main categories: 1) Basic Feature Models, 2) Cardinality Based Feature Models, 3) Extended Feature Models (Metzger & Pohl, 2014).

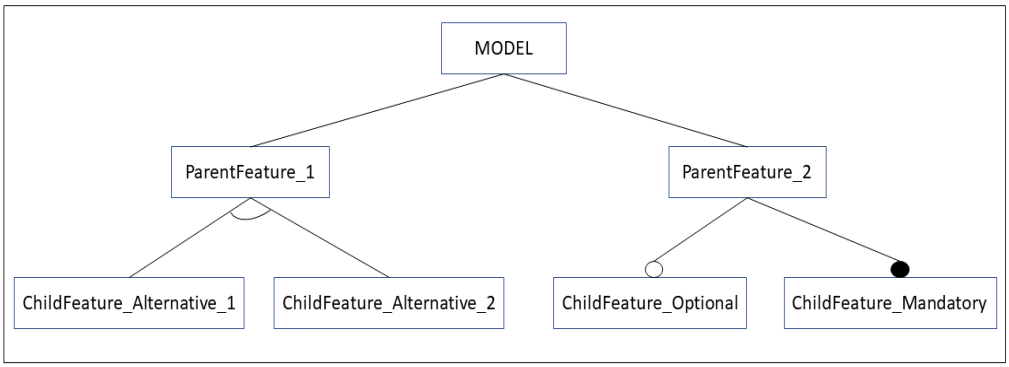

Figure 6 Basic Feature Model Characteristics

Basic feature models offer following characteristics:

- A model is depicted as a feature diagram which is an and/or tree of distinct features. Empty circles represent *optional* features, while *mandatory* features are represented by filled circles. A *parent* feature may have *child* features and *alternative* features are depicted as children of the same parent having a "*one*

and only one can exist" relationship. A child feature can only exist if the parent feature is selected, otherwise child feature is considered *unreachable*. If there is no such relation is given between children, *one or more* child feature can be active (See Figure 6).

- Features can have also interrelationships: *requires* relationship means that one feature needs another feature existence, and *mutually exclusive* relationship means that two features cannot exist at the same time. (Kang, Cohen, Hess, Novak, & Peterson, 1990)

Extended feature modeling technique presents *feature attributes* as the biggest change in modeling approaches until that time. According to Benavides *et al*, attribute of a feature is quantifiable characteristic having a set of possible values. Attributes of distinct features can have interrelationships, and this is called as extra-functional features of a product family (Benavides, Trinidad, & Ruiz-Cortes, 2005).

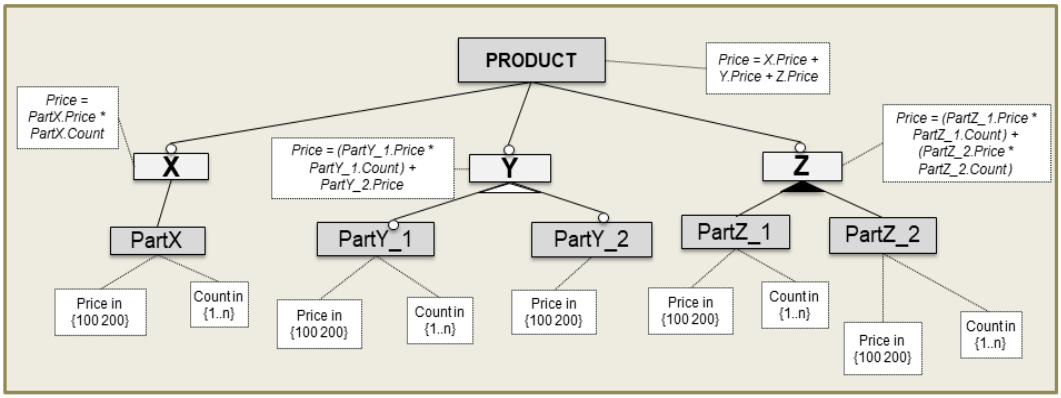

Figure 7 Extended Feature Modeling Example.

Figure 7 demonstrates an extended feature model of a hypothetical product family with X, Y, Z and their possible child features. *Price* and *Count* are attributes of these features. While *Price* attribute can take a value between 100 and 200, *Count* attribute can take value from 1 to n *(attribute domain information*). For such a feature model, followings are an example of extra-functional relationship:

- PartY_1.Count > 5 *requires* PartZ_1
- Y.Price = (PartY_1.Price * PartY_1.Count) + PartY_1.Price

Cardinality based Feature Models had brought to a new concept to variability managing "Cloned Features". This modeling technique defines feature models as follows: A root feature can have *solitary features* or *grouped features.* Solitary features can have cardinality like [0..*] (meaning zero or more times) or [1..n] (meaning x times where $1 \leq x \leq n$). Cardinality indicates that how many times different instances of same feature can coexist on a product. Although the cardinality of some solitary features is not defined explicitly, they have still cardinality implicitly. For example, mandatory features have cardinality of [1..1] and for optional features it is [0..1]. In addition, a solitary feature does not have to have cardinality with a single interval, they may have more than one interval like [1..2][5..7] which means a feature can be cloned by 1,2,5,6,7 times (Czarnecki, Helsen, & Eisenecker, 2005a).

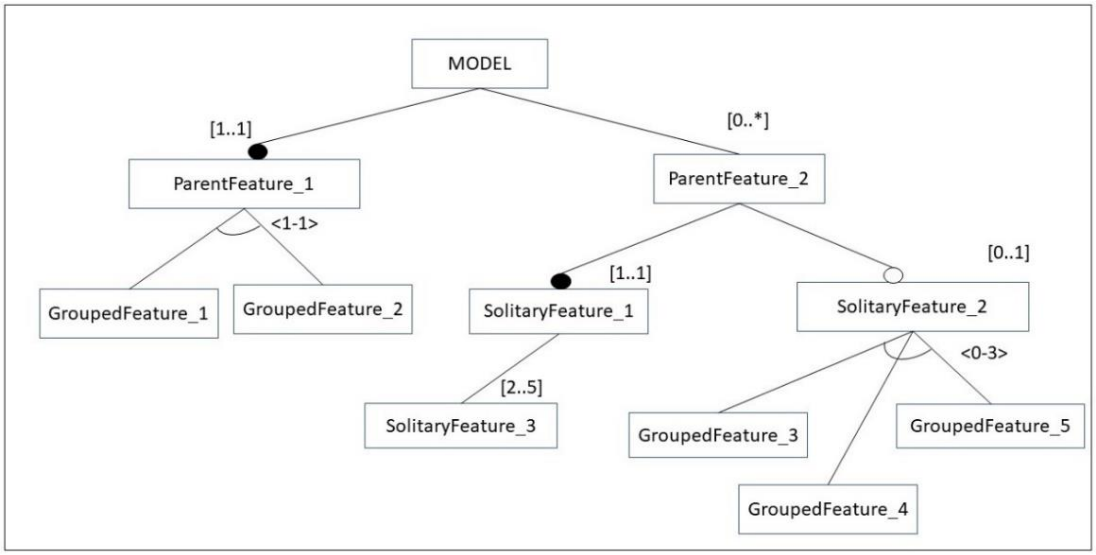

Figure 8 Cardinality Based Feature Model Characteristics

Group cardinality is used for group of k sub features of a parent feature with n lower bound and n' upper bound where $0 \leq n \leq n' \leq k$. For example, for 3 sub features of a parent features, if they have alternative relationship and no explicit cardinality is

given, group cardinality is <1-1>, which means only one sub feature should be selected. However, these sub features can also have cardinality like <0-3> which means 0 or more up to 3 features can be activated at the same time (See Figure 8). According to the Czarnecki et al, features belonging to a group cannot also be a solitary feature because if grouped feature also have solitary cardinality, the bounds defined for the group can be exceeded (Czarnecki, Helsen, & Eisenecker, 2005b).

Feature Models of any type are formed during domain analysis and represent all possible capabilities of product families. To shape one of the end products, a system feature catalog is generated by selecting required features and this catalog identifies the product configuration (Kang, Cohen, Hess, Novak, & Peterson, 1990).

2.3.3. Dynamic Reconfiguration at Runtime Needs

Dynamic needs originating from humans, system or environmental changes for an already running software systems had exposed the need for dynamic adaption. New feature demands for an already developed software product family, customer specific or system specific adaptations have resulted with new approaches and solutions in the literature. While software product line approach aims to create a product portfolio with lower cost and high quality by adopting reusable software components, the emerging need; adaptation to variable environment, sensing the changes in context conditions have led to a newer approach: Dynamic Software Product Lines (Capilla, Trinidad, Bosch, Ruiz-Cort's, & Hinchey, 2014). In today’s large and heterogeneous systems, determining the requirements and features before deployment phase is not enough, instead there is an emerging challenge to handle variability during runtime. DSPL approach supports to define the variability points of a system for both pre and post deployment stages and then suggests feature activation and deactivation mechanisms to handle autonomous decision making and then dynamic reconfiguration of software assets during execution (Cetina, Pelechano, Trinidad, & Cort's, 2008).

Dynamic software product lines provide following properties:

- Adaptation to dynamically varying system (internal) and environment (external) needs.
- Dynamic reconfiguration of product by applying autonomous decision making at runtime.
- Support for wide range of product families with the help of reusable software components.
- Management of variability driven by an explicit variability model.
- Variability management with feature models at run time.

A distinguishing aspect of a DSPL from other dynamically reconfigured systems in general is that a DSPL must possess a variability model explicitly; variability management and dynamic adaptation mechanisms must be using this model. In a product family, the most important thing to maximize the reuse of product components is determination of commonalities and variabilities between them. Every single aspect that is common or distinct can be accepted as a variant for this product family. It is crucial to identify the goals for a system, ask domain questions and give explicit answers to these questions, and in return they are also another variability points/variants (Capilla, Bosch, & Kyo-Chul, 2013). On the other hand, a product also can have some internal states and vary by the environmental changes during execution cycles. To adapt the product to the varying conditions, some features must be activated or deactivated, and the system needs to evolve and achieve self-adaptation. All possible features that can be active for a product is called as "configuration" and *dynamic reconfiguration* of these set of features during runtime is the most important capability for Dynamic Software Product Line Engineering.

In the DSPL context we have a single system (or product) with many possible configurations whereas in traditional SPL we have a product family with many possible products. Note that in the traditional sense, products of a family (SPL) are produced mainly by configuration; in the dynamic sense, appropriate configurations of a system (DSPL) are obtained by reconfiguring it at run time.

Known DSPL Approaches:

a) MADAM

MADAM is an DSPL approach targeting the distributed systems in which modules are connected to each other through a network and continuously reporting their status changes like busyness, network capacity, battery or memory level. (Hallsteinsen, Stav, Solberg, & Floch, 2006). MADAM Approach applies DPSL methodologies and provides an adaptation platform to sense the context changes during runtime. This platform matches the best variants with context changes to reconfigure the system and fulfill adaptation.

MADAM's adaptation platform consists of different parts (See Figure 9). Core module provides services for Component, Resources and Instance Management. Component management provides controlling mechanisms for the product components' deployment for initial time or during run time. Instance Management offers an interface to control the instances of components' life cycle like initialization, attribute value binding, start or stop. Resource management provides monitoring and access mechanism for resources of the system components. Context manager is one of the most important modules of the platform. It always monitors the system, tries to understand the contextual changes that lead to reconfiguration and stores them using Context sensors. These context changes are passed to Adaptation Manager and it examines them if there is a need for reconfiguration. To determine the variants that may be affected by these changes, Planner is used. When the adaptation need is finalized, Configurator reconfigures the application during time.

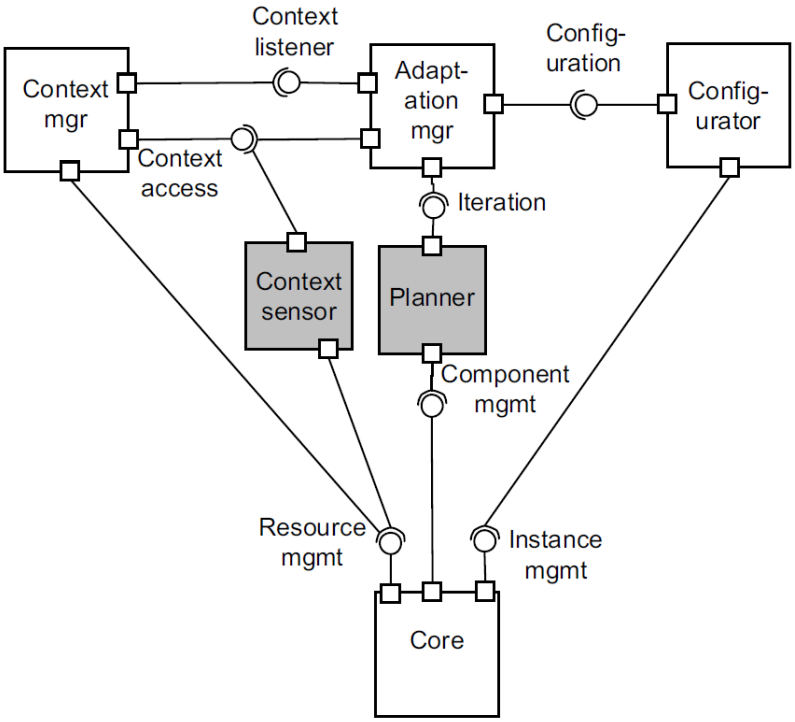

Figure 9 MADAM Adaptation Platform (Hallsteinsen, Stav, Solberg, & Floch, 2006)

b) Model Based Reconfiguration Engine

Model Based Reconfiguration Engine (MoRE) is another approach to capture the context changes during lifecycle of pervasive systems and reflect these changes to current configuration of them. The approach is developed on smart home case study (Cetina, Giner, Fons, & Pelechano, 2009). As a first step, MoRE monitors the changes of devices on smart home and looks for architectural change necessities according to the conditions and corresponding resolutions (configuration). If there is need for a change on the system, reconfiguration plan is prepared. This plan contains actions to adopt current architecture to the needed one and check for consistency between models and resulting configuration is done (See Figure 10). For example, in home detection mechanism of smart home, the system monitors the home continuously and

when the owner leaves the home, movement sensors are activated, and their status are watched continuously through communication channels.

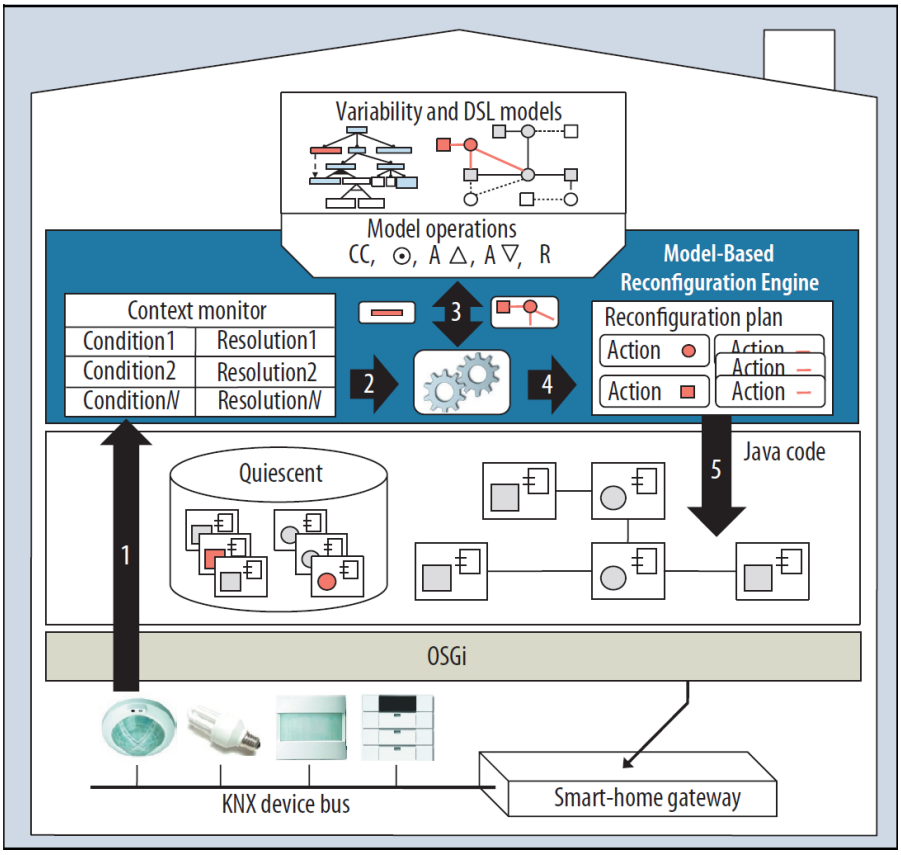

Figure 10 Model Based Reconfiguration Engine (Cetina, Giner, Fons, & Pelechano, 2009)

MoRE is developed on top of OSGi framework and uses JAVA dynamic capabilities to realize the dynamic reconfiguration. It identifies reconfiguration actions under three different group. First group is Component related actions which are activating or deactivating components of system without restarting it. OSGi infrastructure is used to accomplish component install, start, stop or uninstall. Once a component is activated on the system, Channel actions are fulfilled under the second group of actions. Channels actions connect the activated component system by using OSGi Wire class which adopts publish subscribe pattern. Lastly, Model actions are carried out by reflecting architectural changes to feature model of the system (Cetina, Giner, Fons, & Pelechano, 2009).

c) Models@runtime

Models@runtime is an approach to leverage model driven software engineering in such a way that these models can be adapted to changing execution environments. Models@runtime aims to create methodologies to identify adaptation needs of software models at runtime environment and adjust them to these changes without or with very little human intervention (Blair, Bencomo, & B. France, 2009).

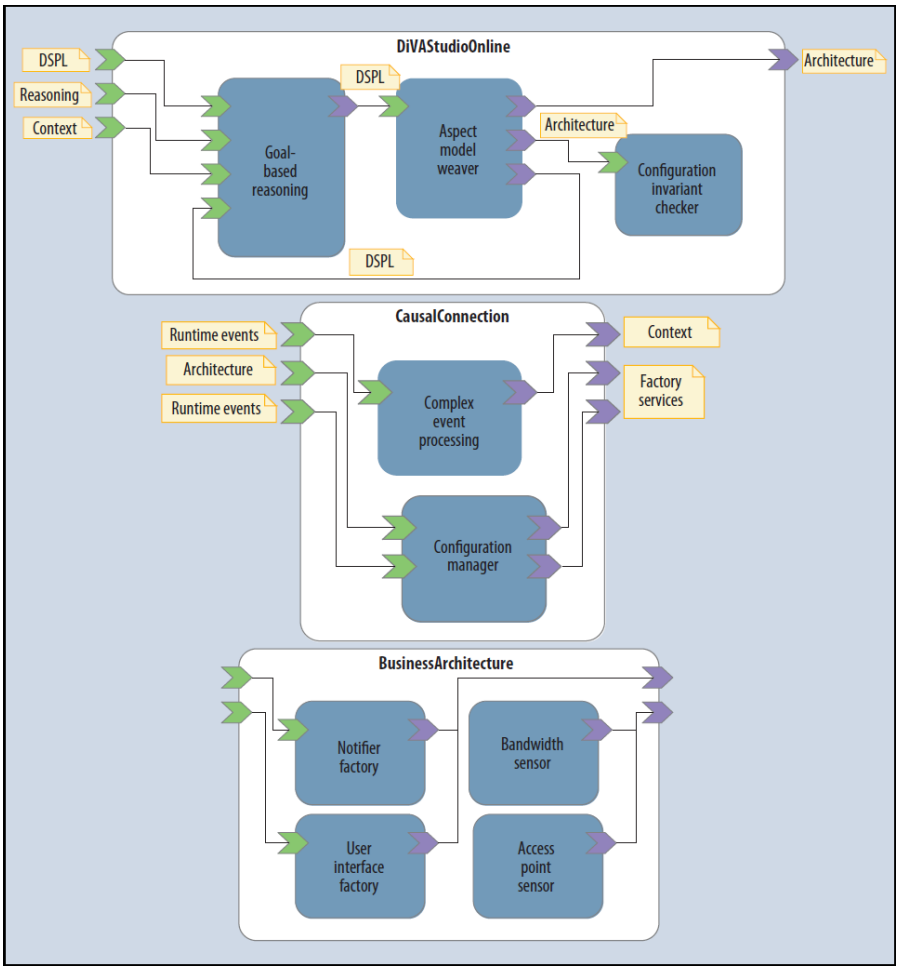

Figure 11 Models@runtime Architecture (Morin, Barais, Jezequel, Fleurey, & Solberg, 2009)

Models@runtime tries to build Dynamically Adaptive Systems (DAS) by identifying goals and behaviors of their components. To build these systems, Models@runtime advises to develop models that is abstract enough to dynamically evolve during execution time of the system. (Morin, Barais, Jezequel, Fleurey, & Solberg, 2009)

Figure 11 depicts the three-layered runtime architecture including DiVAStudioOnline, CasualConnection and BusinessArchitecture. While BusinessArchitecture layer is application specific, CasualConnection layer is platform specific and listens runtime events and updates current context accordingly by also using architecture. Then Goal Based Reasoning Engine determines the feature list to be activated. Aspect Model Weaver of DiVAStudioOnline Layer chooses to corresponding aspects for selected features to generate the resulting architectural model. Configuration Checker is responsible for validating resulting configuration after aspect weaving is completed. If the configuration is valid then Configuration Manager on CasualConnection layer create components, binds them to system and handles active ones also by creating reconfiguration commands (Morin, Barais, Jezequel, Fleurey, & Solberg, 2009)

CHAPTER 3

FEATURE MODEL OF THE REMOTE MONITORING SYSTEM

3.1. Summary

This chapter presents the commonality and variability analysis for our remote monitoring management (RMM) system for computer-based systems. The feature Modeling technique has been chosen to express the variation points of the system. In the first part, feature building by considering problem space and solution space dimensions is discussed. In the following part, the new concept of *Dynamic Feature Attribute* is introduced.

3.2. Feature Modeling of RMM System

In the RMM systems, management involves monitoring of workstations and collecting the monitoring outputs. Therefore, determination of monitors according to the profile of clients is the initial step of system development. Management of clients is out of the scope of this study and after this point, study will be focusing on the Remote Monitoring of workstations, however, the system will continue to be referred to as RMM to ensure consistency. For the Commonality and Variability analysis of RMM, which is the building block of DSPLs, we adopt the aspect and viewpoint-oriented approach introduced by Kyo C. Kang and Hyesun Lee in (Capilla, Bosch, & Kyo-Chul, 2013). C&V analysis of a product line can be modeled in many ways based on different viewpoints adhering to the principle of separation of concerns. In the problem space, user goals and objectives, required quality attributes, and product usage contexts are typically modeled in product line engineering. In the solution space, C&V is modeled for the functional dimension (i.e. capabilities, services), the operating environmental dimension (e.g. operating system, middleware), and the design dimension (e.g. domain technologies).

Problem Space Dimension:

Some variability points are derived from the goals that are addressed by different assets of a software product line. These goals shape the product features and the variability model of product line in return. From this point of view, it is useful to identify following concepts for C&V of RMM systems:

- *The goal:* Enable IT staff with a remote monitoring and management system for a large group of workstations.
- *Usage Context:* Remote Monitoring and Management system will be available for various operating systems and it will be working as a background task, without interrupting the end user. It should enable IT technician to configure RMM for different type of user groups as well as individual users.
- *Performance Attribute:* RMM must be functioning for up to 2000 online clients at a time.

Solution Space Dimension:

- *Capabilities:* Clients can have different profiles in accordance with the several categories such as Resource Utilization, Security, Performance and Energy. According to the profile applied to a client workstation, different monitors are dynamically loaded or unloaded at run time.

- *Operating Environment:* Windows (minimum supporting version: XP)

In the light of above requirements especially solution space dimension - capabilities related ones, a feature dictionary is constructed (See Table 1). This dictionary presents the detailed information about all possible abstract and concrete features, optional and mandatory features and constraints. These features are mapped to only functional requirements of RMM system. Non-functional requirements including operational environment related ones are out of scope and will be provided as future work.

Table 1 Feature Dictionary

Feature	Definition
Energy	This *optional* feature is the last of four main abstract features of the RMM. It has one child feature: Idle State Monitor. (See Figure 13)
Idle State Monitor	It is a concrete feature that monitors the idle state of the system. It continuously checks the indicators of idle state; whether the screen protector is active, an input user is present, or the mouse and the keyboard are not used for a certain period. It reports the idle state changes to RMM. It has only common "*ID"* attribute (See 3.3)
Security	This feature is the second of four main abstract features of the RMM. This *optional* feature aims to detect security vulnerabilities on computer systems. It has three child features: Access Restriction, Malware Detection, Intrusion Detection. (See Figure 14)
Access Restriction	It is an abstract feature to watch computer drivers with restricted access to drivers. New Device Blocked and CD/DVD Detection Monitors are concrete children. When Access Restriction is activated, one of these features must also be activated.
New Device Blocked Monitor	This feature continuously monitors new device blocking events reported by antivirus programs. It has only common "*ID"* attribute (See 3.3) If antivirus detects and blocks a new device, it is reported to RMM immediately.
CD/DVD Detection Monitor	This feature continuously monitors the CD/DVD drive. It has only common "*ID"* attribute (See 3.3) For example, if a DVD is inserted to computer, it is reported to RMM.
Malware Detection	This is an abstract feature to monitor the malware attacks to computer systems. When Security feature is activated, this feature must also be activated. Two concrete child features; Malware Handled and Unknown Application Running (UAR) Monitors can be alternatively active in RMM.
Malware Handled Monitor	This concrete feature monitors the malware handled events generated by antivirus programs and reports the issue to RMM immediately. It has only common "*ID"* attribute (See 3.3)
Unknown Application Running Monitor	This concrete feature monitors unknown applications that has an expired certificate or no certificate and reports the security vulnerability to RMM. Event Log Monitor must also be activated

	when this feature is activated. It has only common "*ID"* attribute (See 3.3)
Intrusion Detection Monitor	This feature is an abstract feature to detect the unauthorized entry to the system. Intrusion Detection Monitor must be activated when Security feature is activated. At least one of its five concrete children, namely, Firewall Actions, Network bandwidth, TCP, Ping and HIPS Monitors, must also be activated.
Firewall Actions Monitor	This concrete feature monitors the firewall actions. Network Bandwidth Monitor is required when this feature is activated. It contains following attributes other than common "*ID"* attribute (See 3.3): • *"Action Type"* attribute (Blocked, Allowed, Asked and Blocked or Asked and Allowed) • "*Rule*" attribute containing "*Number Value"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if number of actions with given *action type* reaches the *number value* within a given *interval*, this value and action type is reported to RMM.
Network Bandwidth Monitor	This feature monitors the network bandwidth. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Rule*" attribute containing "*Threshold for Network Bandwidth Usage"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if *network bandwidth usage* becomes *greater than* the *threshold* value during the given *interval*, network bandwidth usage is reported to RMM immediately.
Ping Monitor	This feature monitors a given host via ping method. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Rule*" attribute containing "*Host Name"* and a "*Ping Condition"* (up or down) • "*Interval"* attribute used for duration for monitoring. For example, if a given *host* (URL) is *down* for given *interval*, the packet loss is reported to RMM immediately.
TCP Monitor	This feature monitors the specified TCP Port of host at the specified frequency. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Rule*" attribute containing "*Host Name", "Port"* and a "*TCP Status"* (open or closed) • "*Interval"* attribute used for duration for monitoring.

	For example, if requested *port* of a *host* is *closed* during the given *interval*, it is reported to RMM together with the status.
Host Intrusion Prevention System (HIPS) Events Monitor	This feature monitors the HIPS events generated by antivirus program. It contains following attributes other than common "*ID"* attribute (See 3.3): • *"Intrusion Type"* attribute (Blocked, Allowed, Asked and Blocked or Asked and Allowed) • "*Rule*" attribute containing "*Number Value"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if number of HIPS events with given *intrusion type* reaches the *number value* within a given *interval*, this value and intrusion type is reported to RMM as a vulnerability.
Performance	This **mandatory** feature is the second of four main and abstract features of the RMM system. It has four child features monitoring the components of computer systems that can be a measure of performance: CPU, Disk Space, RAM and Event Log Monitors. These features are coexisting and at least one of them must be active in RMM (See Figure 15).
CPU Monitor	This feature watches the percentage of current CPU usage on the workstation. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Rule*" attribute containing "*Threshold for CPU Usage"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if CPU Usage becomes *greater than* the *threshold value* during the given *interval,* CPU usage is reported to RMM immediately.
Disk Space Monitor	This feature keeps track of the usage of disk space on the workstation. It contains following attributes other than common "*ID"* attribute (See 3.3): • *"Disk Name"* attribute used to identify the disk to be monitored • "*Rule*" attribute containing "*Threshold for Left Disk Space"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if Disk space becomes *less than* the *threshold value* during the given *interval,* left disk space is reported to RMM immediately.

RAM Monitor	This feature keeps track of RAM usage on the workstation. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Rule*" attribute containing "*Threshold for RAM Usage"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if RAM Usage becomes *greater than* the *threshold value* during the given *interval*, RAM usage is reported to RMM immediately.
Event Log Monitor	This feature monitors the Event Log component of Windows Operating System. It contains following attributes other than common "*ID"* attribute (See 3.3): • *"Event ID"* attribute indicating an identifier for events given by Windows • "*Rule*" attribute containing "*Event Level ID"* and "*Event Source"* • "*Interval"* attribute used for duration for monitoring. For example, if an event with *ID* 26 and *source* WindowsUpdateClient is generated, this event is reported to RMM immediately.
Resource Utilization	This **mandatory** feature is the first of four main abstract features of the RMM. It has 4 different child features watching the different sources of computer systems: Folder Size, File Size, Process and Service Monitors. Resource Utilization feature is must for RMM, and the system must have at least one child feature of it (See Figure 16).
Process Monitor	This feature continuously watches a predefined process. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Process Name*" attribute identifying the process to be watched. • "*Process Status*" attribute identifying monitoring condition (on or off). For example, if the state of given process becomes *on/off*, status is reported to RMM immediately.
Service Monitor	This feature continuously watches a predefined service. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Service Name*" attribute identifying the Windows service to be watched. • "*Service Status*" attribute identifying monitoring condition (on or off).

	For example, if the state of given service becomes *on/off*, status is reported to RMM immediately.
Folder Size Monitor	This feature continuously watches the size of the predefined folder. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*Folder Path*" used to identify folder to be monitored. • "*Rule*" attribute containing "*Threshold for Folder Size"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring For example, if the size of folder becomes *greater than* the *threshold value* during the given *interval*, folder size is reported to RMM immediately.
File Size Monitor	This feature continuously watches the size of predefined file. It contains following attributes other than common "*ID"* attribute (See 3.3): • "*File Path*" used to identify file to be monitored. • "*Rule*" attribute containing "*Threshold for Folder Size"* and a "*Relational Operator"* (greater than, equals, etc.) • "*Interval"* attribute used for duration for monitoring. For example, if the size of file becomes *greater than* the *threshold value* during the given *interval*, file size is reported to RMM immediately.

Figure 12 shows the Remote Monitoring System Feature Model by using the Extended Feature Modeling approach (Benavides, Trinidad, & Ruiz-Cortes, 2005). It depicts the cross-tree relationships between the monitors defined on dictionary and the attributes of these features. The model and dictionary reflect only functional dimension of common and variable points belonging to RMM system, and other dimensions are out of the scope of this thesis. Every monitor depicted in the model represents a feature of the RMM system. The activation and deactivation of these features determine the current configuration of the system. For some product lines there may exist potentially conflicting or dependent features and RMM system modeled in this study also has these kinds of features.

Following are interrelationships between the features:

- When Firewall Actions Monitor is chosen by IT manager for this client, Network Bandwidth Monitor is *required* and must also be activated to control the injections.
- Because Malware affections means that possibly workstation user is online and there is an active network traffic over computer, RMM system *excludes* the Idle State Monitor when Malware Detection Monitor is active.
- Unknown Application Running Monitor *requires* Event Log Monitoring because this monitor needs event information created by applications.

Cloned Features of RMM:

RMM system allows IT managers to create same monitor with different rule definitions for multiple times and to deploy all of them to the same workstation. For example, IT manager can define two Service Monitor with following rules:

Service Name*: wuaserv*	**Service Status:** *On*
Service Name: *Aruva Service*	**Service Status:** *Off*

To create such monitors, RMM system allows cloneable features. Every feature has an ID attribute and this attribute indicates the number of each clone of the same monitor. Therefore, it can be said that RMM Feature Model adopts also the Cardinality Based Feature Modeling approach. As it is shown in the Figure 12, other than Event Log Monitor, the number of clones is not limited with a certain number. This number can be defined during application engineering phase by the implementer. Therefore, only Event Log Monitor cardinality is depicted on the model.

*RMM system feature model also has two **constrains**:*

- While child feature cardinality is 0 to n, Event Log Monitor can exist up to 8 instances at the same time.

- For a selected configuration, total CPU usage of features cannot exceed 70%.

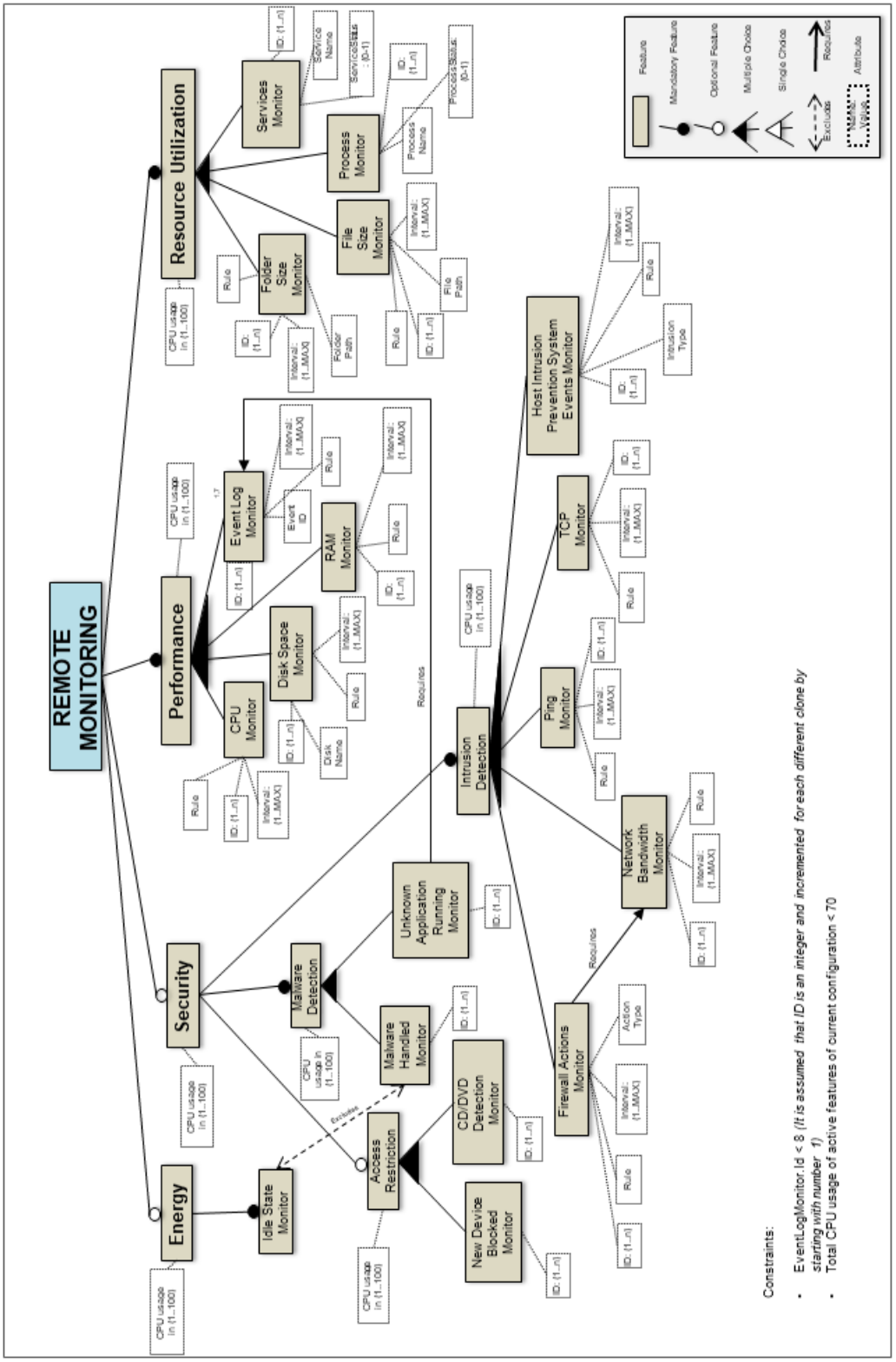

Figure 12 RMM Feature Model

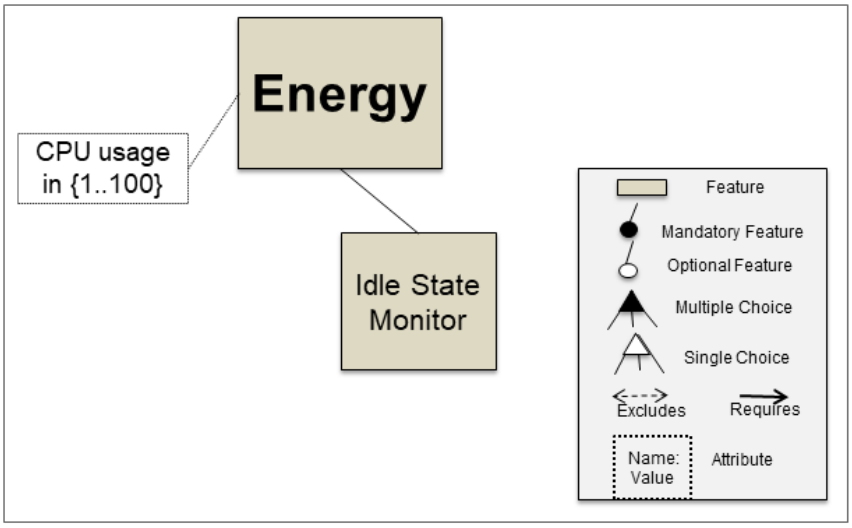

Figure 13 Energy Subtree of RMM Feature Model

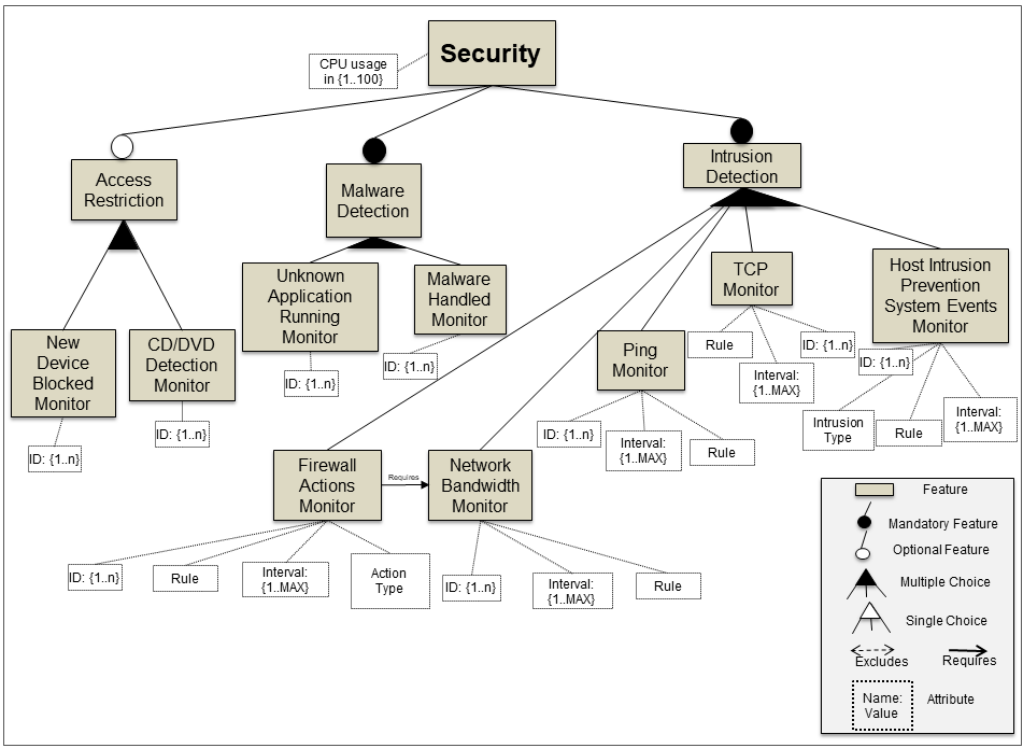

Figure 14 Security Subtree of RMM Feature Model

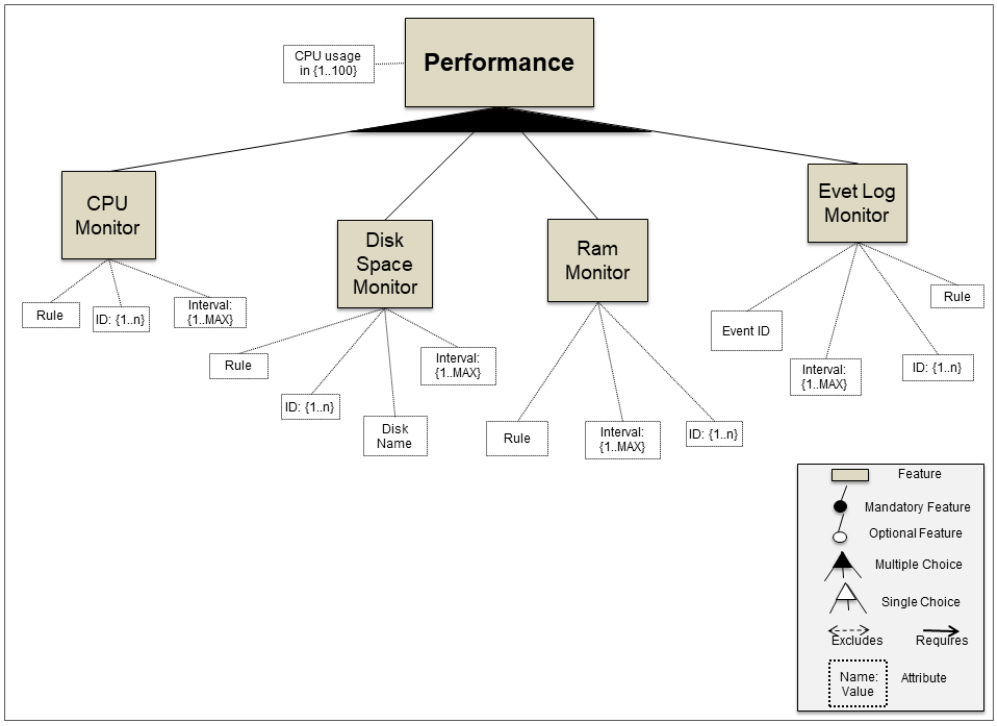

Figure 15 Performance Subtree of RMM Feature Model

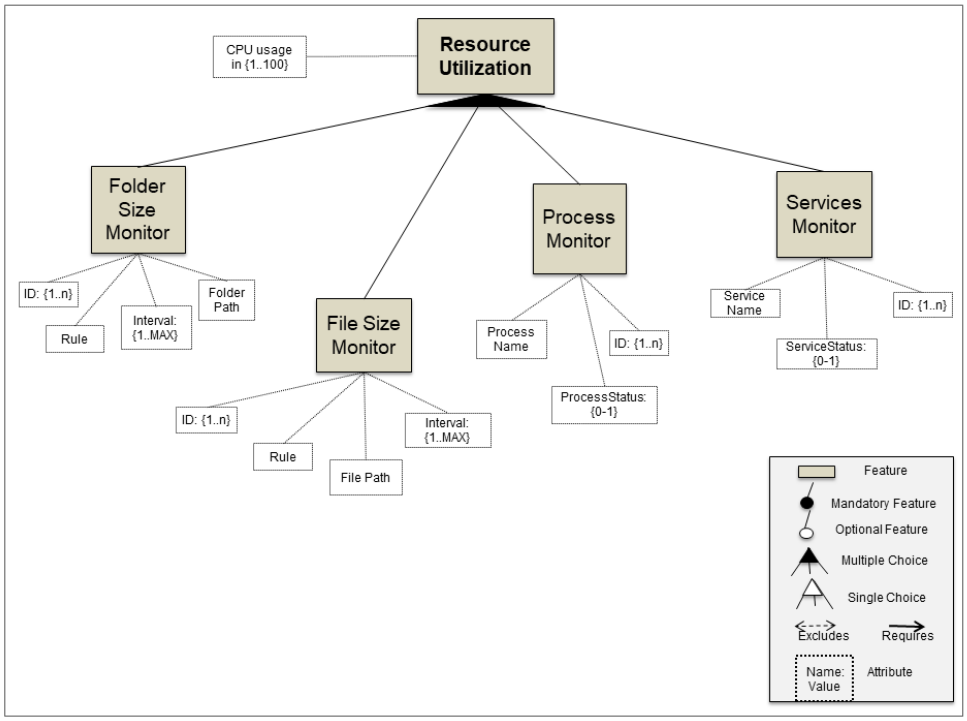

Figure 16 Resource Utilization Subtree of RMM Feature Model

3.3. Feature attributes and Extra Functional Features

Feature Model of the RMM system defined on this study adopts Extended Feature Modeling technique which provides *attribute* and *extra-functional feature* definitions. Attributes and these extra functional feature definitions had been given to express the quality related requirements of product families (Benavides, Trinidad, & Ruiz-Cortes, 2005).

Common Attributes:

- **ID:** Every child feature has *ID* attribute taking value from 1 to n indicating feature identifier. As it is stated before on this chapter, its value is incremented sequentially for every clone of the monitor. “n” can be defined on application engineering / development phase.

- **Interval**: This attribute is used for duration and taking value between 1 to MAX which is defined during application development.

- **Rule:** This attribute is shared by most of the child features. *Rule* attribute identifies the monitoring details, for example Folder Size Monitor Rule contains *Threshold for Folder Size* and *Relational Operator* (greater than, equals, etc.) parameters.

Table 1 gives the specifications for the other attributes specific to child features. The restrictions defined on **constrains** previously on this chapter, but it is better to explain them in detail. Our model has following extra-functional features:

- Event Log Monitor.ID < 8: meaning there can exist maximum 7 Event Log Monitor at a time.

- Parent Feature CPU usage = Sum of Child Features:

  e.g. Malware Detection.CPU Usage = malware Handled Monitor.CPU Usage + UAR Monitor.CPU Usage

- Total CPU Usage of active features must not exceed 70 percentage.

3.4. A New Concept: Dynamic Feature Attribute

All feature attributes mentioned on this study is static and their values can be determined before deployment time except one attribute*: CPU Usage*. Current attribute definition given by Benavides et al meets the requirement of assigning a static value to attributes during domain or application engineering phases, or even at deployment time, but for *CPU Usage* attribute, it is nearly impossible to determine before execution of product. The measurement of CPU Usage of an active monitoring may be affected by computers' hardware specifications, operating system or other instant running applications. The same monitor's CPU Usage can even vary on the same computer during its life cycle.

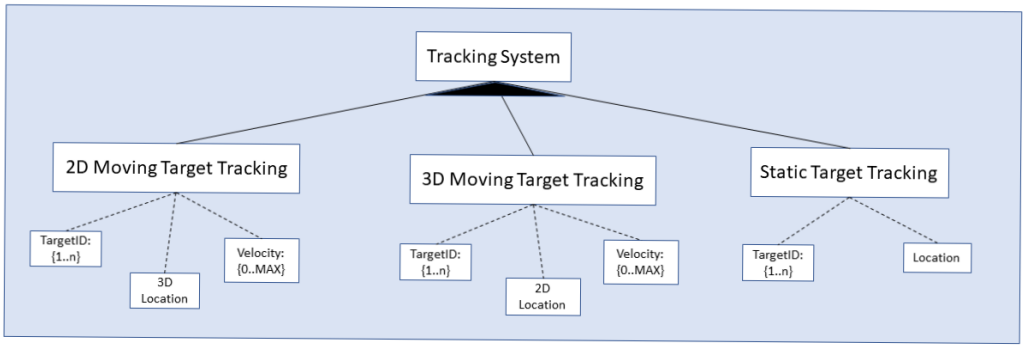

Figure 17 Tracking System Feature Model with Dynamic Attributes

Target Tracking system, whose feature model is given Figure 17 can be shown as another example. This system is a like turret carrying a weapon to track and then hit a specific target. It can follow a static target or moving target with 2D and 3D planes. When 2D moving target tracking feature is activated, the system needs target ID, its location on 2D plane and its velocity. TargetID attribute is a static attribute defined early by the user operating the system. However, 2D location and velocity attributes cannot be defined statically before application deployment time since they are

dynamic and change over time. These variable values are continuously measured by a different system having thermal cameras and fed to the system in real time.

Such feature attributes whose value could only be determined during execution time and changing during feature's life cycle can be named as *Dynamic Feature Attributes*. Such attributes seem more suitable to use on Dynamic Software Product Lines, because DSPLs address to the system whose requirements, components and use cases can vary during the life cycle of the product. The core of DSPL Engineering is understanding these runtime changes during the execution time and adopt the product accordingly. Therefore, dynamic attributes can be regarded as a component changes that must be understood during runtime and the system can be reconfigured according to its value.

Challenges About Dynamic Feature Models

Unlike static feature attributes whose values can be set during application engineering phase of SPL Engineering, dynamic attributes cannot be statically set to a value before deployment of product. While, this characteristic of dynamic feature attribute is good for nonfunctional requirements with varying values, it also leads to some drawbacks.

- *Global Constraint Problem:* Feature models can have global constraints affected by the attribute values of activated features. For example, RMM Feature Model has a global constraint as follows: *"*Total CPU Usage of active features must not exceed 70 percentage*"*. Because CPU usage measurement is changing over time for lifetime of a monitor, it is challenging to calculate total CPU Usage of activated monitors.

- *Extra Functional Feature Problem:* Parent feature attribute values can be calculated by sum of children attribute values as an example of extra functional features. However, dynamic attributes also cause problem for such a calculation because dynamically changing values of children feature attributes leads to varying values for parent feature attributes.

This study contributes to DSPL methodology with a new concept *Dynamic Feature Attributes* definition but presented study will not manage them for the challenges stated above. Instead, these attributes will not be considered during framework design and will not be handled on the case study implementation. Dynamic feature attribute managing during the life cycle of DSPL products will be presented as future work.

CHAPTER 4

DETAILED DESIGN OF DYNAMIC SOFTWARE PRODUCT LINE

4.1. Summary

This chapter presents the detailed design for Dynamic Software Product Line of Remote Monitoring of Computer Systems. While first part explains the architectural design and implemented classes and packages particularly, the second part enlightens the adoption of context awareness and dynamic binding mechanisms.

4.2. Architecture

Remote Monitoring application as IT Security Management tool consists of two parts; one of which is the *Portal* assisting IT personnel / MSPs to create and manage monitors and the other is the *Agent* controlling client workstations according to the profiles assigned to them. The DSPL presented in this study provides an infrastructure to implement the ***Agent*** component of Remote Monitoring systems and implementation of ITSM portal is out of scope for this study.

In Remote Monitoring and Management Systems, varying *Profiles* are created and deployed to set of client workstations to control and manage them remotely by MSPs. For example, for a group of users who must work under high security, an MSP can form a *Profile* including following monitors:

- From Security Related Monitors: *Malware Handled Moni*tor, *Firewall Actions Monitor*, *HIPS Event Monitor*
- From Performance Related Monitors: *CPU Monitor* with 70% threshold, *RAM Monitor* with 70% threshold with 5 min. duration.

Each monitor defined on this profile corresponds to a child feature on RMM Feature Model. Therefore, a profile can be directly mapped to a configuration including set of active features. This close relationship facilitates the determination of the features corresponding to the monitors in the profile and the activation of them dynamically during runtime. DSPL for RMM Systems adopts the component based architectural style. Monitors are designed as *cloneable* and reusable software components referred from RMM feature model.

Figure 18 shows the use of the system during the general flow of the remote monitoring. Flow starts with the profile deployment by MSP (Step 1), then needed configuration -set of features to be activated- is interpreted according to the new profile (Step 2). The new configuration must be checked by Configuration Manager to ensure that it conforms to the feature model (Step 3). After this verification, the monitoring components inferred from the features are set up (Step 4) and corresponding Qt plugins (The Qt Company, 2018) are loaded at runtime (Step 5). After that point plugins start to monitor the workstation with respect to the rules defined by MSP (Step 6).

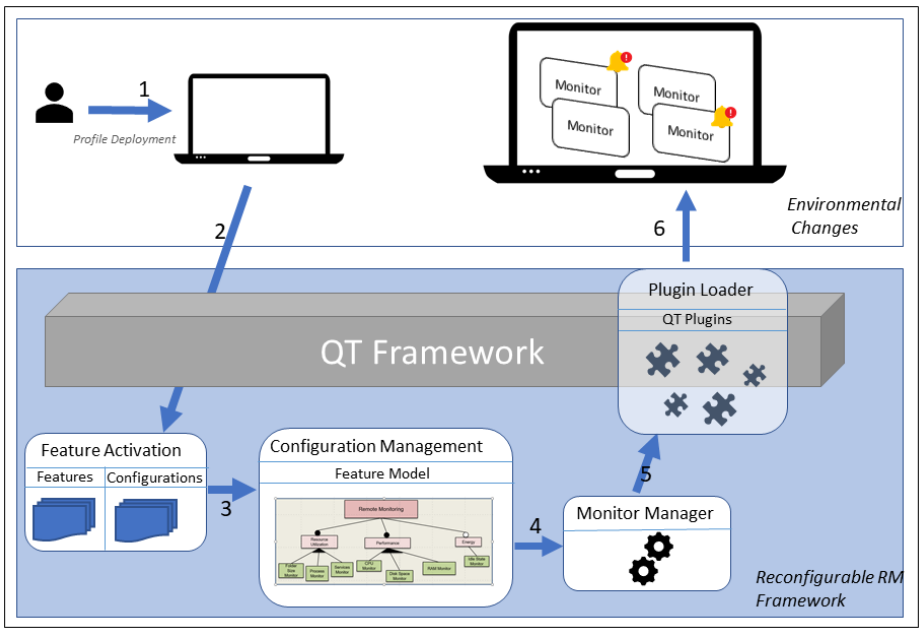

Figure 18 Component Based Architecture of DSPL for RMM of Computer Systems

4.2.1. Core Classes – Plugins

The monitor is realized as a dynamically reconfigurable component on the system. When there is need for a new feature activation, corresponding component is loaded at runtime. As stated on 3.2, RMM Feature Model has cloned features; therefore, the monitors are also designed as cloneable and a new monitor instance whose attributes are bound to different values can be deployed by several times. To provide this functionality, Factory Design Pattern is used (See Figure 19). *Every monitor plugin is implemented as a factory to create multiple instances of same monitor*. Therefore "Monitor – Plugin – Factory" trio is used commonly for the class and interface naming.

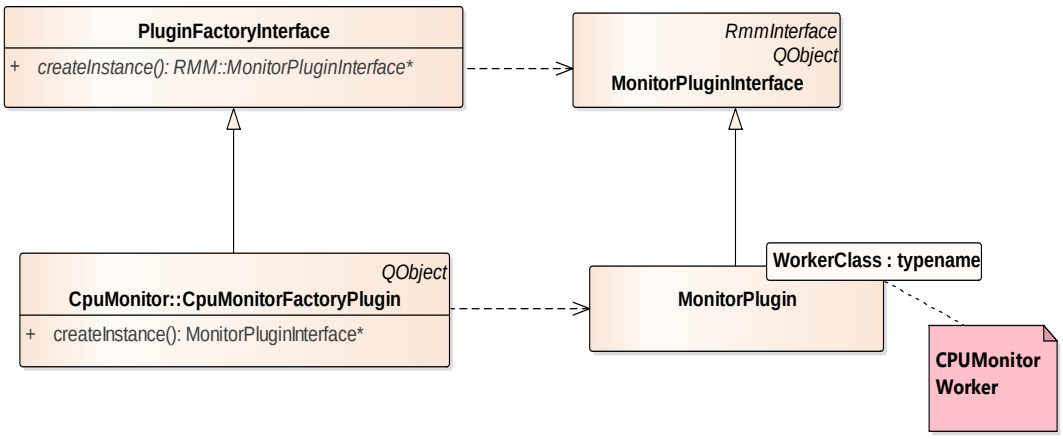

Figure 19 Plugin Factory Design

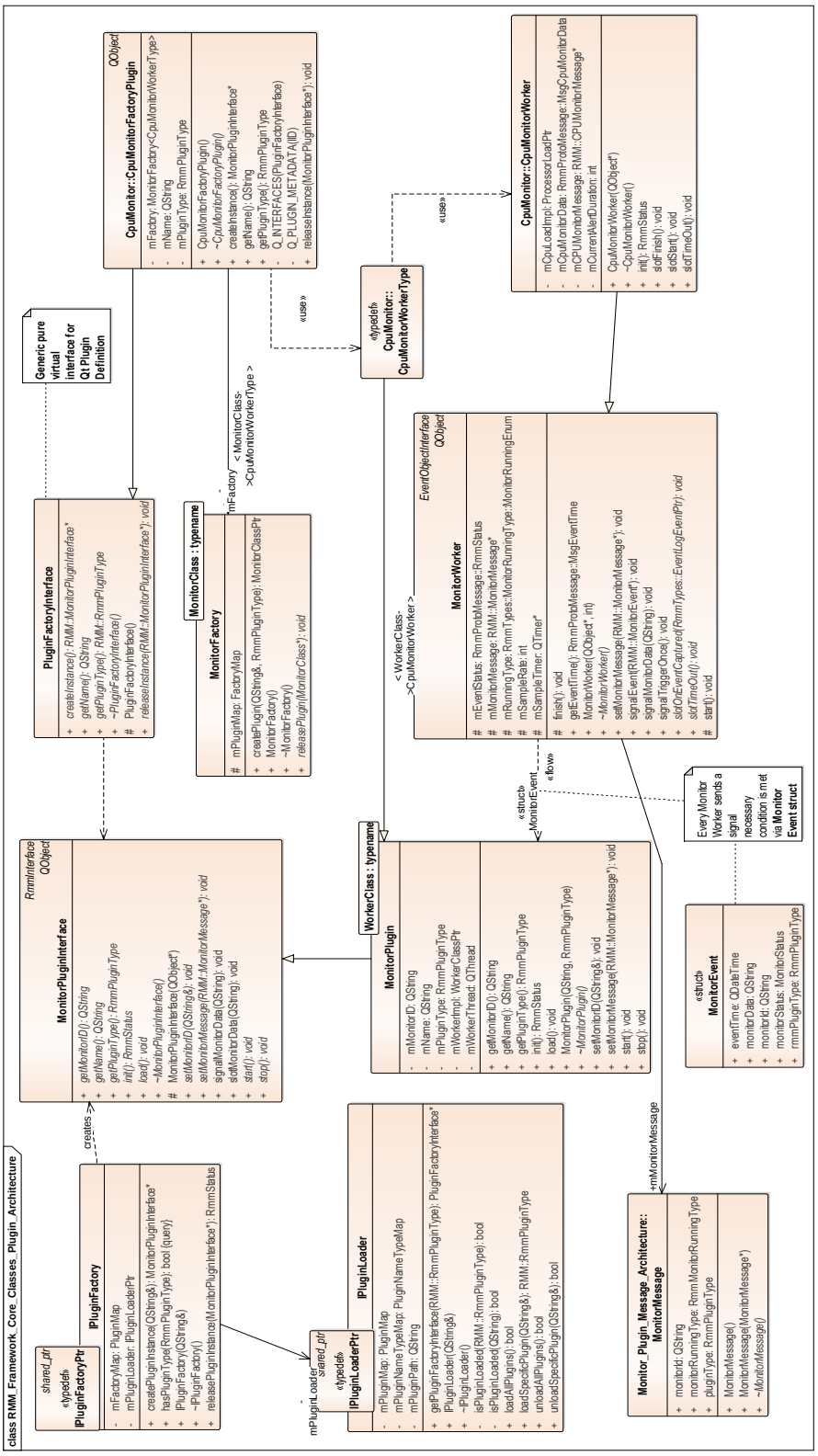

Figure 20 Core Classes - Plugins

Following core classes are presented by the study to implement dynamic loading mechanism and dynamic reconfigurability on runtime in return, with the help of extended Qt Application by “The Low-Level API” methodology (The Qt Company, The Low-Level API: Extending Qt Applications, 2018). They are also depicted on Figure 20 as UML class diagram.

- **IPluginLoader,** is the class whose responsibility is to provide Qt Plugin loading and unloading mechanisms during runtime. It needs a folder path in which plugin dll files are located as constructor argument. Single plugin with a given name or bulk loading / unloading functionalities are available thanks to QPluginLoader class (The Qt Company, QPluginLoader Class, 2018).

- **IPluginFactory,** is a class used as a factory to create an instance of a Monitor Plugin Factory with a given “name” by using IPluginLoader class. It also has a capability to release an already created instance by iterating through the plugin map saved before.

- **PluginFactoryInterface,** is an interface with pure virtual functions, implemented by plugins to be defined and loaded dynamically (See Figure 20). This interface is needed to extend a Qt Application to detect and load plugins during runtime with the help of QPluginLoader. This interface also informs Qt meta object system about the plugin interface with following command: *Q_DECLARE_INTERFACE(PluginFactoryInterface, PLUGIN_INTERFACE_RUID)*

- **MonitorPluginInterface,** is an interface implemented by MonitorPlugins. The type of instances created by monitor factory plugins is MonitorPluginInterface (See Figure 20).

- **MonitorPlugin,** is a template class with template parameter WorkerType. This class has a member “mWorkerImpl” with type of WorkerType. It moves

mWorkerImpl qobject to a new qthread, starts it, transmits worker messages to upper layers through Qt Signals and Slots mechanism (The Qt Company, Signals & Slots, 2018), and stops it when it is necessary.

- **MonitorFactory,** is a template class with template parameter MonitorClass. It is designed in such a way that, every specific plugin; for example, CPU Monitor Plugin, will use this class by realizing MonitorClass as a **MonitorPlugin** to fabricate cloned CPU Monitors. MonitorPlugin instances are also bounded to a specific worker implementation, for example monitoring CPU usage.

- **MonitorWorker,** is parent class for different monitoring worker implementations. It contains common members for monitor status and monitor rule.

4.2.2. Core Classes – Relationships

Every Qt plug-in generated by the DSPL is a monitor factory to create multiple monitors of same feature, for example 3 CPU monitors with different thresholds. To create a Qt Plugin, a generic interface with pure virtual functions as stated at Figure 21 should be provided. **PluginFactoryInterface** is presented for this purpose. This interface provides generic functions to create multiple instances of corresponding monitor. Every specific monitor factory like CpuMonitorFactoryPlugin, RamMonitorFactoryPlugin, FileSizeMonitorFactoryPlugin etc. is derived from PluginFactoryInterface**.**

```
#ifndef PluginFactoryInterface_h__
#define PluginFactoryInterface_h

#include "MonitorPluginInterface.h"

#include <QtPlugin>

class PluginFactoryInterface
{
public:
        virtual ~PluginFactoryInterface()
        {}
        virtual QString getName() = 0;
        virtual RMM::RmmPluginType getPluginType() = 0;
        virtual RMM::MonitorPluginInterface* createInstance() = 0;
        virtual void releaseInstance(RMM::MonitorPluginInterface* instance) = 0;

protected:
        PluginFactoryInterface() = default;
};

Q_DECLARE_INTERFACE(PluginFactoryInterface,
PLUGIN_INTERFACE_RUID)
```

Figure 21 PluginFactoryInterface Definition

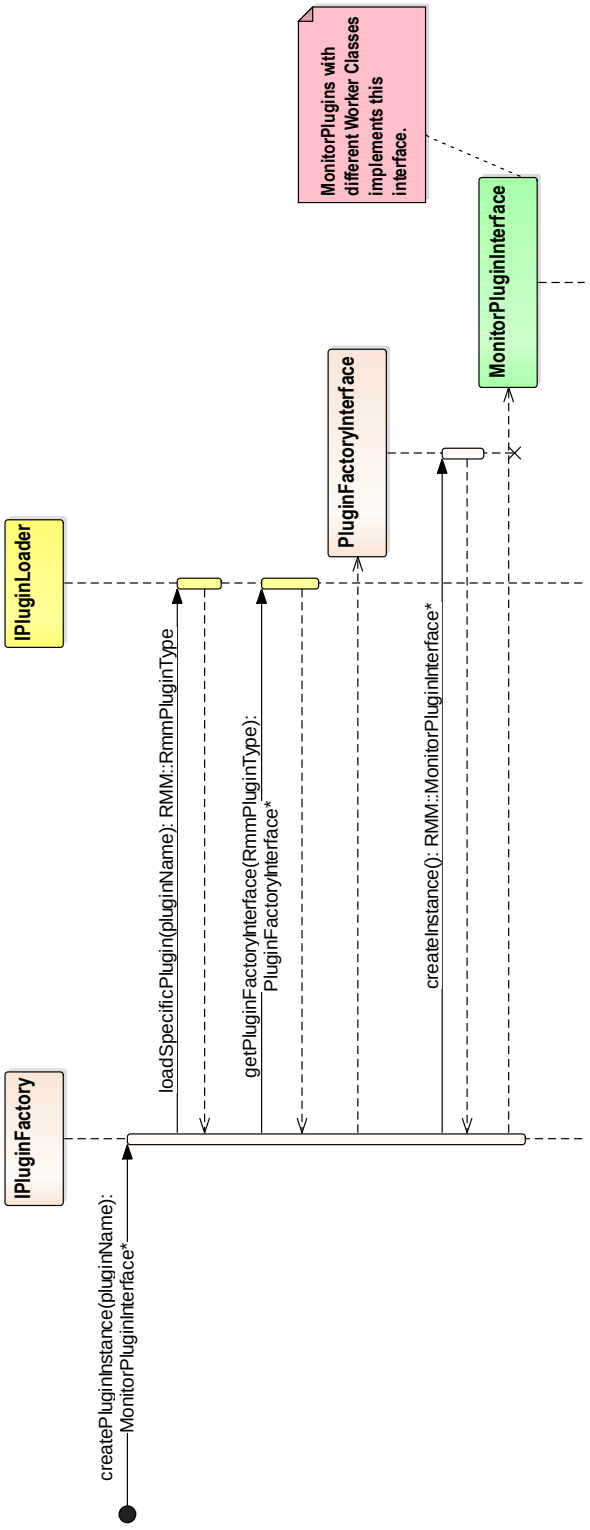

Figure 22 Dynamic Plugin Loading During Runtime

Figure 22 depicts Qt Plugin loading mechanism used to load monitor factory plugins independent from monitor type. For any type of monitor, IPluginFactory class' *createPluginInstance* function is called with the *pluginName* parameter, and this parameter is determined according to feature to be activated. IPluginFactory uses IPluginLoader class instance to load the Qt plugin to memory. After load is successfully done, **PluginFactoryInterface** instance is obtained. This object is then used to create monitor instances with corresponding rule definitions.

Every specific factory plugin definition implementing PluginFactoryInterface should also have a mFactory member with type of MonitorFactory**. MonitorFactory** is a template class with typename **MonitorClass**. MonitorClass typename is replaced with **MonitorPlugin** class at runtime to create different MonitorPlugins corresponding to features. MonitorPlugin class is designed as template class with typename **WorkerClass**. WorkerClass is bound to different workers like CPUMonitorWorker, RAMMonitorWorker, etc. It means that different MonitorPlugins are created by different workers bindings. For example, CPUMonitorFactoryPlugin (which is a Qt plugin implementing PluginFactoryInterface) has a member variable mFactory in type of **MonitorFactory<MonitorPlugin<CpuMonitorWorker>>**. All implementation details are given during the implementations of workers, like **CpuMonitorWorker**.

This mechanism eases the new plugin writing process for application engineers, because all other components like PluginFactoryInterface, MonitorFactory and MonitorPlugin classes are reusable during the new plugin implementation. Only worker specification is enough to create a new plugin.

After a monitor factory plugin is loaded to memory, then *createInstance* function is called on this instance (See Figure 22). This function fabricates an object with the given monitor type, which is derived from MonitorPluginInterface. For example, when CPUMonitorFactoryPlugin is loaded dynamically, we have a factory to generate a cloneable MonitorPlugin with CPU type. *createInstance* function defined on

PluginFactoryInterface is called from this loaded plugin, factory creates an instance implementing **MonitorPluginInterface**.

MonitorPluginInterface provides application developers with an interface to init, start or stop plugins. After start function is called, WorkerClass is moved to new QThread and starts to monitor the computer asynchronously according to the given rules. This process is essence of "*dynamic plugin loading at runtime*" infrastructure, which enables application implementers with a *dynamic reconfiguration* mechanism. After factory plugins for monitors corresponding the possible feature set are created, they are ready to deploy to computer systems and can be plugged and played.

Monitor factory plugins implemented as Qt Plugins can also **be unloaded during runtime without restarting the applications**. The unload mechanism provided by the framework is presented on Figure 23. To release a loaded factory plugin, IPluginFactory class' *releasePluginInstance* function is called with plugin type parameter. IPluginFactory uses IPluginLoader's *loadSpecificPlugin* and *getPluginFactoryInterface* functions to obtain the PluginFactoryInterface pointer for the loaded factory plugin. After then, *relaseInstance* function is used to unload a Qt plugin from memory. An application developer should make sure that all monitor instances are killed before the unload of monitor factory plugin, because IPluginFactory and IPluginLoader do not control this condition while unloading process.

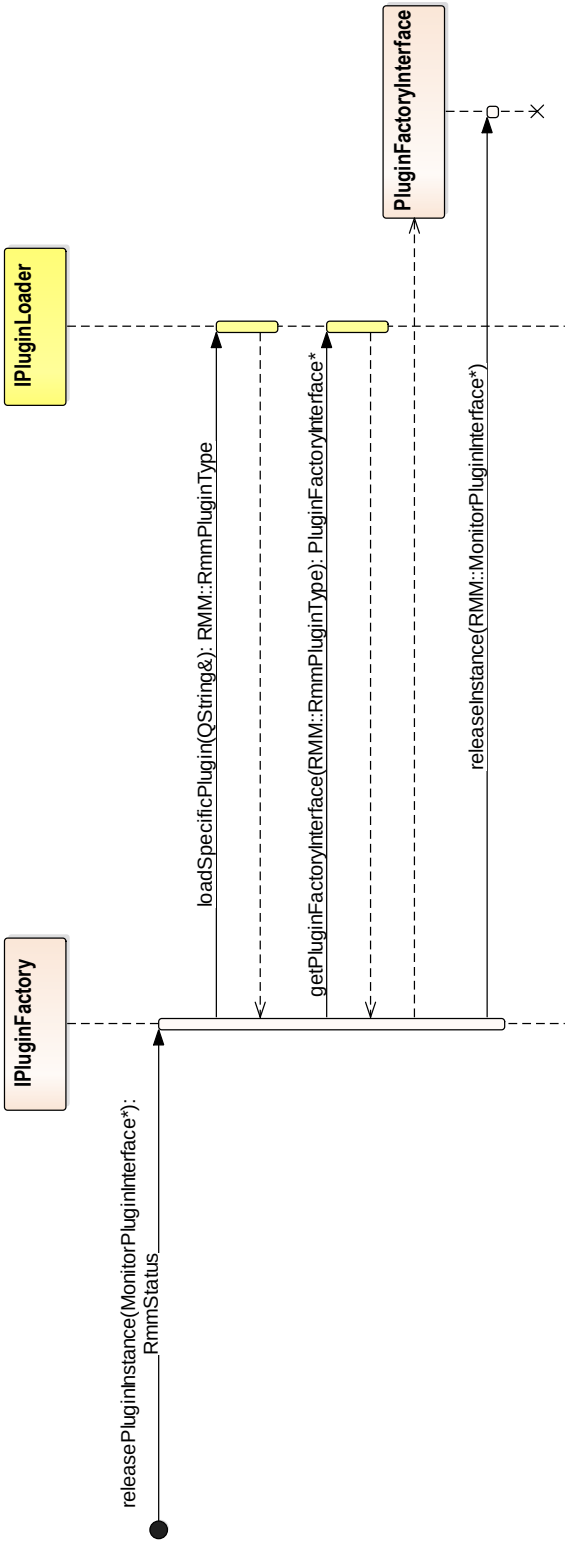

Figure 23 Dynamic Plugin Unloading During Runtime

4.3. Context Awareness and Dynamic Binding Mechanisms

The set of active features make up the current configuration of RMM system. As it is described on Figure 24, new profile deployment or modification on the already deployed profiles are the *triggers* to create a new configuration and consequently a set of active features. Whenever new profile is gathered by the RMM agent of client workstation during execution of the system, reconfiguration is needed. However, after new set of active features are determined, configuration validation by considering the feature model of the product must be fulfilled. Besides that, during the silent execution, some conditions that may affect the states of the features arise. For example, an IT manager can define a profile in such a way that, whenever NetworkBandwidth threshold exceeds 80% for 15 minutes, start FirewallActions Monitor. This is another trigger for RMM system to start reconfiguration process, however it is planned to be added as a future work.

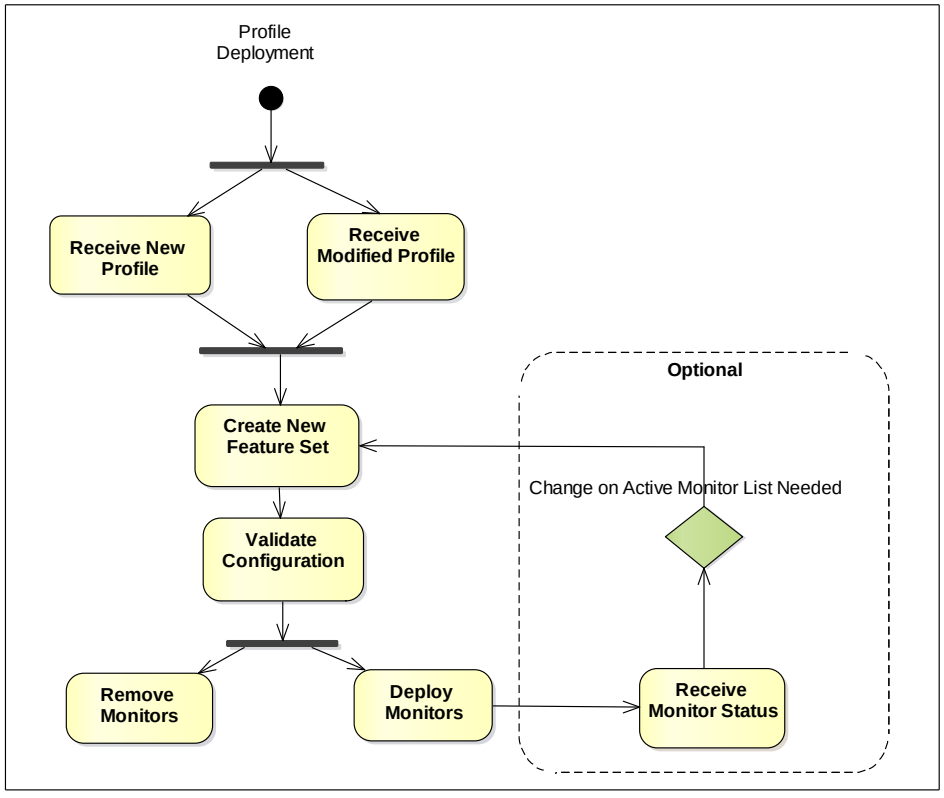

Figure 24 RMM Activities

Our DSPL provides infrastructure to integrate reusable factory plugin components to the product, as context changes occur during the runtime. Context changes in RMM system means changes on profiles assigned to user groups as stated before. Sensing such changes and deploying necessary monitoring mechanisms on the fly are called *context awareness* and *dynamic binding* at runtime.

Figure 25 illustrates the IFeaturActivator, ConfigurationValidator, IPluginFactory and IPluginLoader class relationships that are used for feature activation and deactivation during runtime.

IFeatureActivator, is the component of the DSPL which is responsible to determine the features that must be activated or deactivated conforming to the changes in profile or client workstation modifications (results of the monitors as stated in the previous example). It *dynamically binds the necessary configuration to the running system* by passing a set of features to ConfigurationValidator component. This behavior of the system is called as **context awareness** and **self-adaptation** on post deployment phase of the product.

ConfigurationValidator, is the component that uses a feature model to validate the state of product that will be passed. This component uses a validation algorithm (Entekhabi, Karataş, & Oğuztüzün, October 2018), as a black box. Set of features that is wanted to be activated, and deactivated are passed to ConfigurationValidator, along with the current configuration. If validator returns success, it also provides only set of features that must be activated. If it returns failure, an empty set is provided. According to the result provided by ConfigurationValidator, IFeatureActivator activates the necessary features. Figure 26 illustrates the feature activation process in detail. Plugins corresponding to the features to be activated are loaded to memory by using Dynamic Plugin Loading mechanism. Then monitor instances are created and their workers are started to monitor the computer. This mechanism is an example of dynamic binding features at runtime.

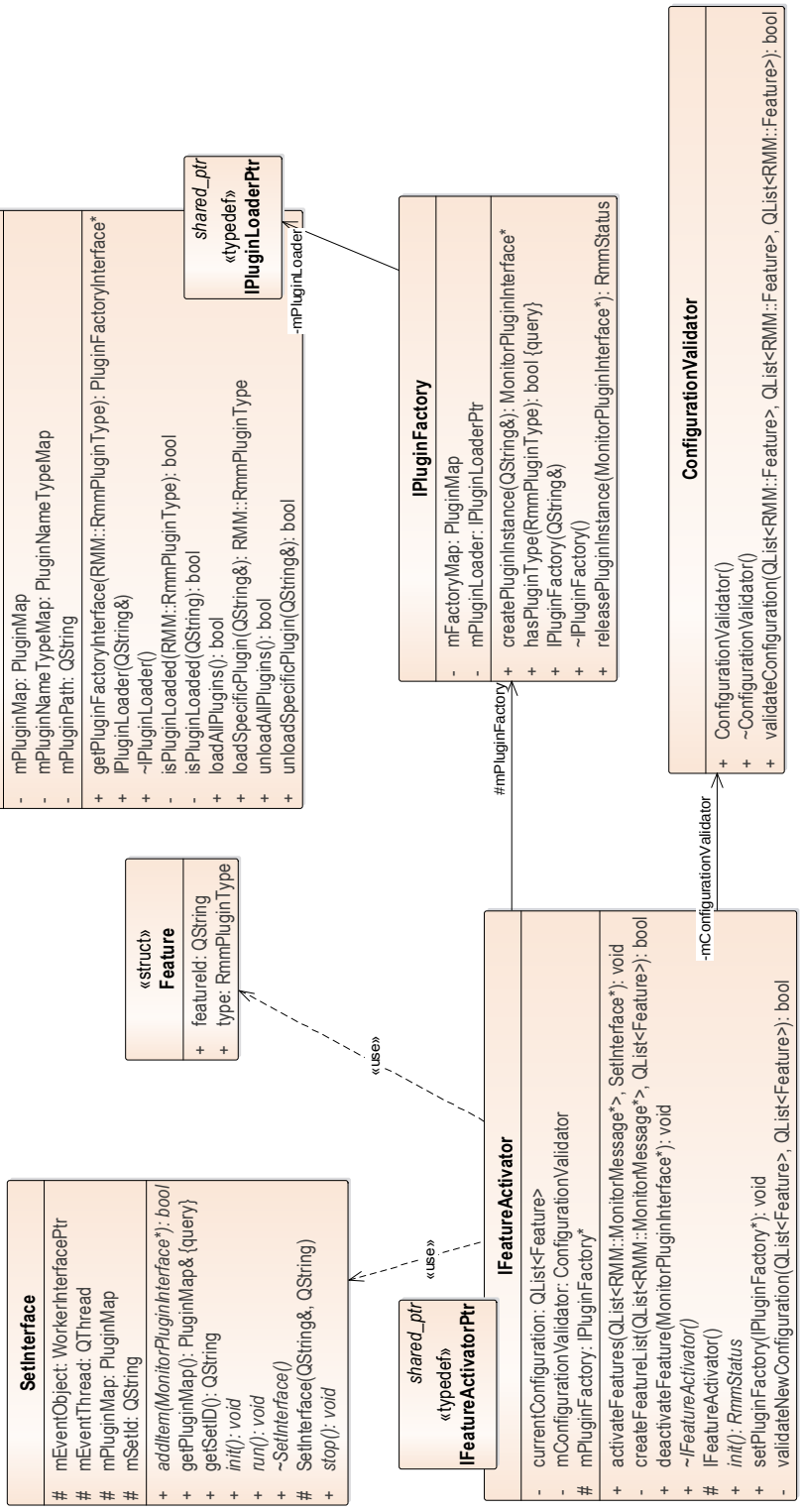

Figure 25 Core Classes - Feature Activation

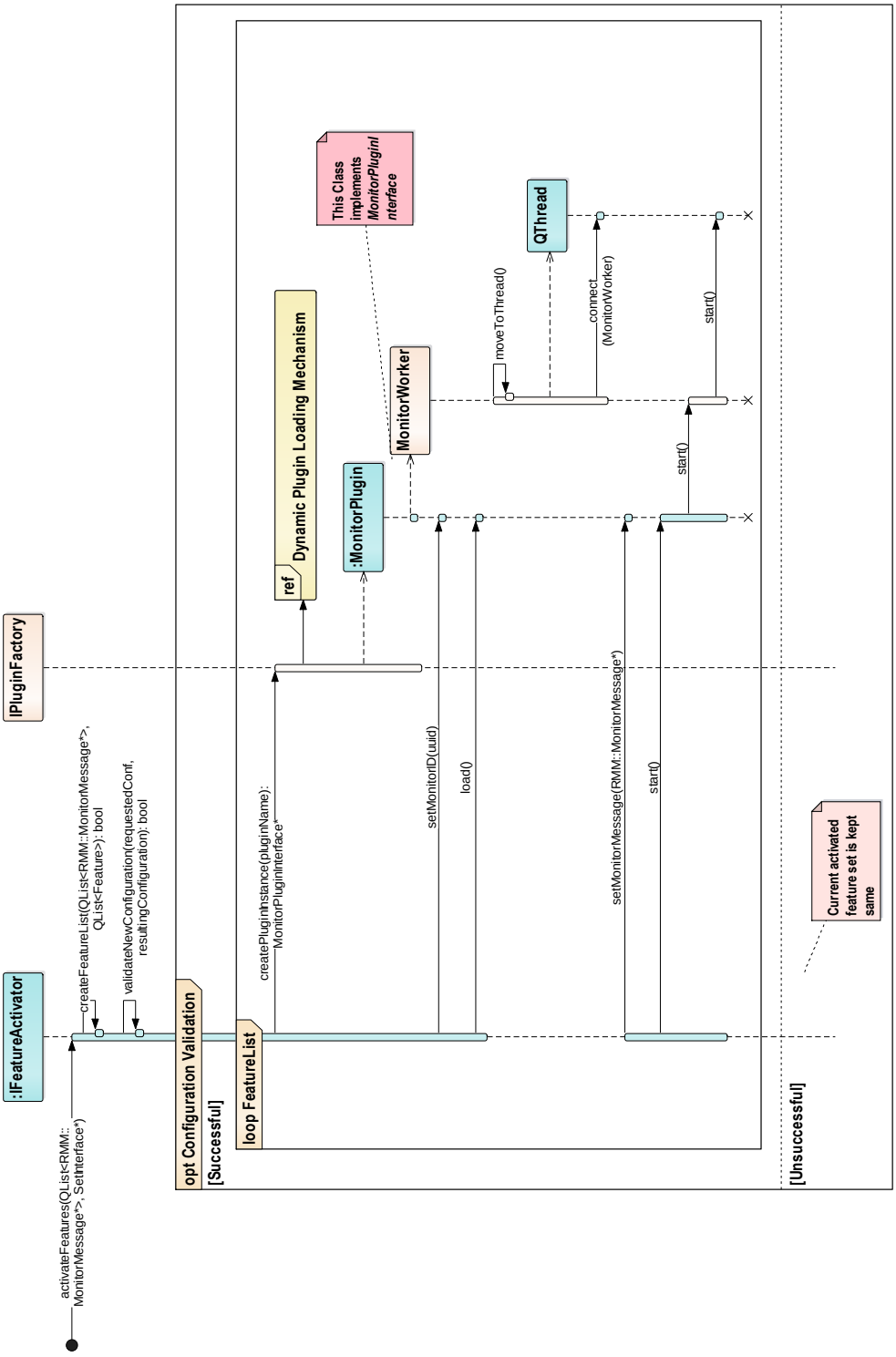

Figure 26 Feature Activation & Configuration Validation

CHAPTER 5

CASE STUDY: COMODE ONE ITSM PRODUCT

5.1. Summary

This chapter introduces the Comodo ONE ITSM tool and the minimal RMM agent developed with the help of the infrastructure provided by the Dynamic Software Product Line for Remote Monitoring of Computer Systems study. Product Overview subsection gives the details about ITSM tool and the specific RMM product along with its feature set and its configurations. Following subsection explains the development of plugins corresponding to the feature set and RMM Agent end to end execution are presented.

5.2. Product Overview

Comodo ONE (C1) Product is ITSM solution for the Managed Service Providers to control many client workstations remotely with the help of automated monitoring and scheduling mechanisms. Its main components are Remote Controller, Automation Library and Scripting, Operating System Patch Management and Remote Monitoring and Management Tools (Comodo Group, 2019).

C1 product aims to automate the IT operations by using predefined procedures, set of scripts and rules. Remote Controller tool is also provided to IT managers, if they need to connect the client workstations and touch their configurations specifically.

As a case study for DSPL for Remote Monitoring of Computer Systems, a minimal RMM product as a part of C1 ITSM Tool is developed by reusing the components provided by the framework. Figure 27 depicts the feature model of the C1 RMM Product derived from the extended Feature Model given at Figure 12, by selecting only Performance and Resource Utilization parent features.

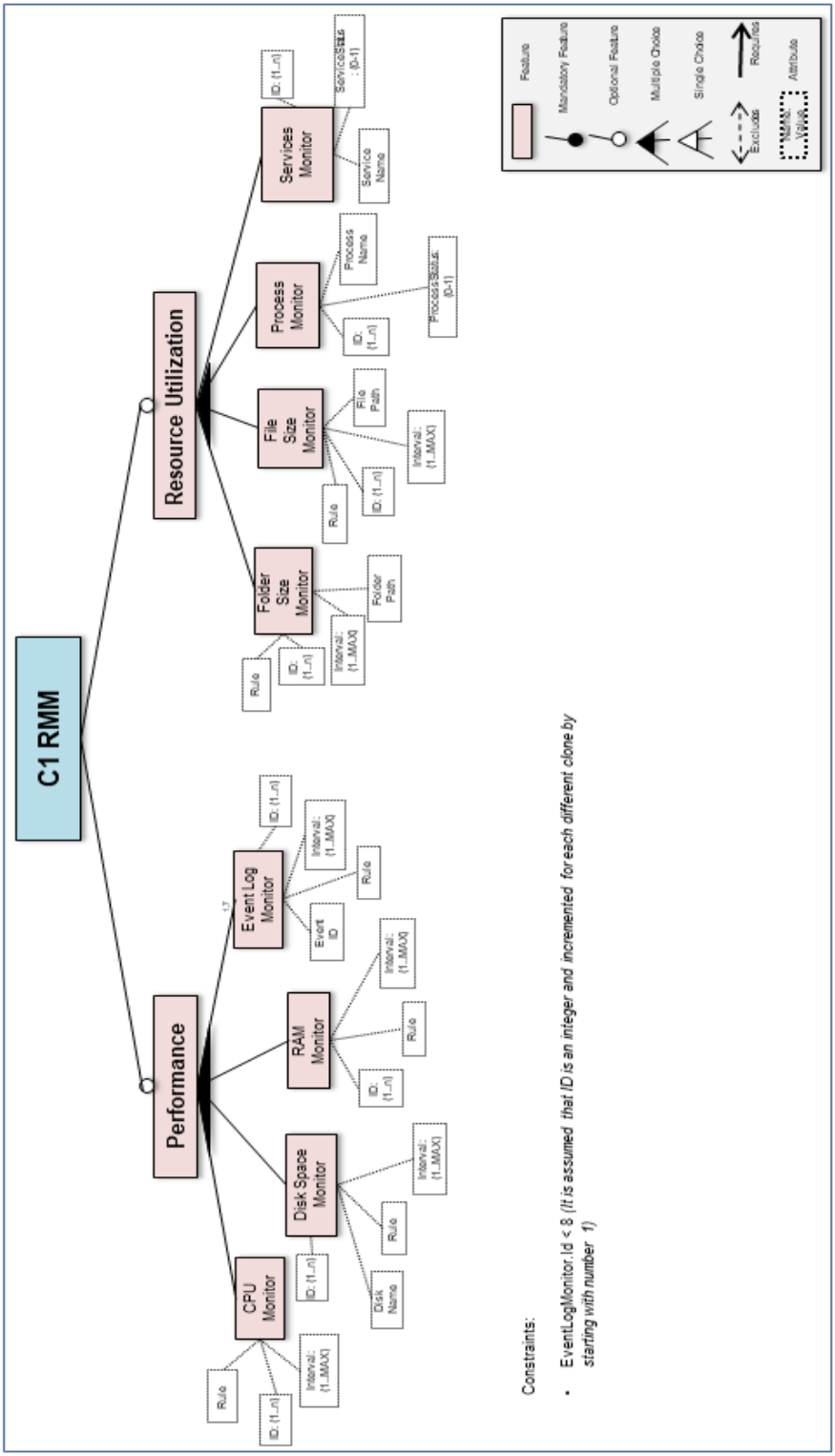

Figure 27 C1 RMM Product Feature Model

Followings are the valid specifications for minimal RMM Product and its features:

- CPU, Disk Space, RAM, Event Log, Folder Size, File Size, Process and Service Monitors are available for users.

- All Monitors are cloneable meaning more than one monitor with the same type can be alive.

- Only Event Log Monitor cannot be cloned more than 8 times. As it is stated before on this study, Windows Operating System has restriction about the number of channels to fetch the data from event log system. It is the only constraint for the product.

- While Performance and Resource Utilization parent features are defined as mandatory on section 3.1 Feature Modeling of RMM System, minimal RMM product developed for this case study have these features as optional. The reason is that MSPs or IT stuff should create their default profile including monitors. Otherwise, RMM agent should activate some monitors corresponding to mandatory features.

- IT Manager can deploy more than one profile to same set of users, modify or delete existing profiles without restarting the system. These profiles can have intersecting monitor sets.

- Minimal RMM agent will respond to profile changes during runtime, by understanding the changes over profiles, binding them to features, and activating or deactivating corresponding monitors.

CPU Usage attribute is removed from FM of minimal RMM product because it is also not handled by the framework as it is stated before.

5.3. RMM Agent Built with DSPL

Table 2 includes created plugins implementing PluginFactoryInterface for minimal RMM agent.

Table 2 C1 Minimal RMM Agent Plugins

Performance Related	Resource Utilization Related
CPUMonitorFactoryPlugin	FolderSizeMonitorFactoryPlugin
DiskSpaceMonitorFactoryPlugin	FileSizeMonitorFactoryPlugin
RAMMonitorFactoryPlugin	ProcessMonitorFactoryPlugin
EventLogMonitorFactoryPlugin	ServiceMonitorFactoryPlugin

As a need for a Qt application extended through plugins, followings have been fulfilled by every plugin of minimal RMM agent (The Qt Company, The Low-Level API: Extending Qt Applications, 2018):

- Every plugin inherits from QObject Class
- Every plugin is registered to Qt meta-object system about its interface (PluginFactoryInterface) by using Q_INTERFACES() macro.
- Every plugin is exported by using Q_PLUGIN_METADATA() macro.
- Every plugin is tested by qobject_cast() function whether it implements the given interface (PluginFactoryInterface)

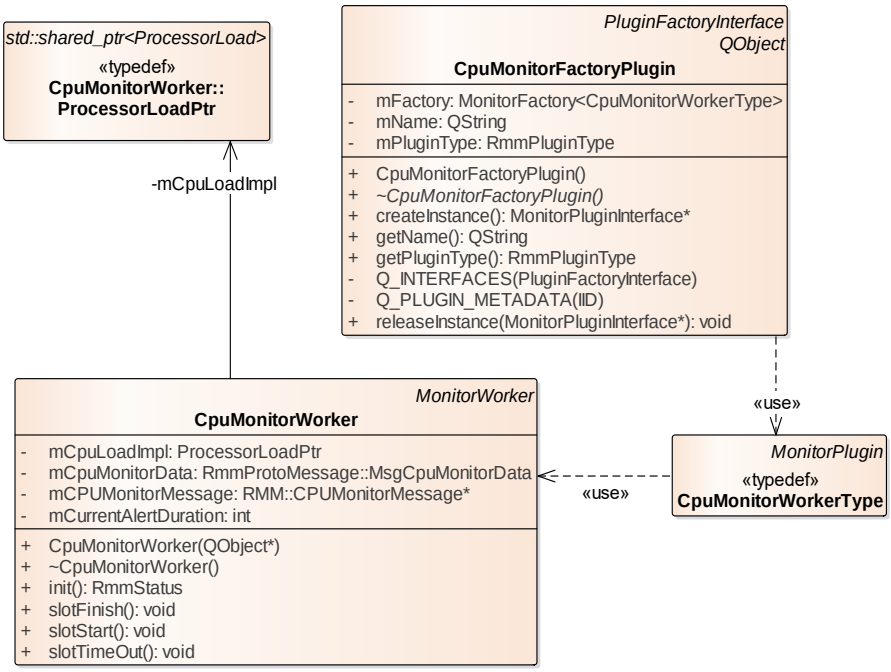

Figure 28 CPU Monitor Plugin Classes

As an example plugin, CPU Monitor class and their relationships are depicted on Figure 28. Besides that, Figure 29 demonstrates the CpuMonitorFactoryPlugin class definition in detail. This class implements PluginFactoryInterface as need for being a Qt Plugin. Besides that CpuMonitorFactoryPlugin overrides the functions coming from PluginFactoryInterface interface, this class is also marked as QObject (Q_OBJECT declaration) and as Qt Plugin (Q_PLUGIN_METADATA(IID PLUGIN_INTERFACE_RUID) declaration). As it is stated before on Chapter 4, all plugins are designed as factories to create multiple instances of the same monitor. Therefore, once CpuMonitorFactoryPlugin as a Qt Plugin is loaded to memory during runtime, createInstance() function is used to populate instances of CPU Monitor Plugins.

```
#ifndef CpuMonitorFactoryPlugin_h__
#define CpuMonitorFactoryPlugin_h__

#include "core/PluginFactoryInterface.h"
#include "core/MonitorFactory.h"
#include "CpuMonitorWorker.h"

namespace RMM
{
	class CpuMonitorFactoryPlugin : public QObject, public
PluginFactoryInterface
	{
		Q_OBJECT
		Q_PLUGIN_METADATA(IID PLUGIN_INTERFACE_RUID)
		Q_INTERFACES(PluginFactoryInterface)

	public:
		CpuMonitorFactoryPlugin();
		virtual ~CpuMonitorFactoryPlugin();

		QString getName() override;

		RmmPluginType getPluginType() override;

		MonitorPluginInterface* createInstance() override;

		void releaseInstance(MonitorPluginInterface* instance) override;

	private:
		QString mName;
		RmmPluginType mPluginType;

		MonitorFactory<MonitorPlugin<CpuMonitorWorker>> mFactory;
	};
}
#endif // CpuMonitorFactoryPlugin_h__
```

Figure 29 CpuMonitorFactoryPlugin Class Definition

All MonitorFactoryPlugin classes have a factory instance called *mFactory* which is specialized according to the monitor type. This specialization is provided through worker classes and its interface is given at Figure 30. CpuMonitorWorker is derived from MonitorWorker base class which is provided by the framework. This worker class contains the necessary functions to measure the CPU usage overtime according to the given internals. If given conditions defined on the "*Rule*" attribute are met, then this worker class sends the results to main application with the help of Qt Signals & Slots Mechanism. Main application does not take the result into consideration to start a reconfiguration process and it just passes them as logs to ITSM Servers. Then IT Managers/MSP can follow up the monitoring logs by using ITSM Portal.

Other than plugin implementations, IFeatureActivator and ConfigurationValidator classes are directly used during the implementation of minimal RMM Agent.

```
#ifndef CpuMonitorWorker_h__
#define CpuMonitorWorker_h__

#include "core/MonitorWorker.h"
#include "core/MonitorPlugin.h"

namespace RMM
{
	class CpuMonitorWorker : public MonitorWorker
	{
		Q_OBJECT
		class ProcessorLoad;
		typedef std::shared_ptr<ProcessorLoad> ProcessorLoadPtr;
	public:
		CpuMonitorWorker(QObject *parent = nullptr);
		~CpuMonitorWorker();

		RmmStatus init() override;

		protected slots:

		void slotTimeOut() override;

		void slotStart() override;

		void slotFinish() override;

	private:
		ProcessorLoadPtr mCpuLoadImpl;
		RMM::CPUMonitorMessage* mCPUMonitorMessage;
		RmmProtoMessage::MsgCpuMonitorData mCpuMonitorData;
		int mCurrentAlertDuration;
	};

	typedef MonitorPlugin<CpuMonitorWorker> CpuMonitorWorkerType;
}
#endif // CpuMonitorWorker_h
```

Figure 30 CpuMonitorWorker Class Definition

CHAPTER 6

CONCLUSION AND FUTURE WORK

DSPL for Remote Monitoring of Computer Systems offers a framework for IT service management through remote monitoring of computers by providing dynamically reconfigurable, reusable software components. IT management needs are inevitable and nearly vital for many companies to sustain their product line healthy. Performing ITSM within the companies properly improves efficiency and quality of the work. It is also undeniable that a healthy business development environment also leads to lower costs and reliable products.

Running a remote monitoring system requires monitoring assets to be started and stopped on the fly, independent from main application running state. It means that the user of the system needs a mechanism to deploy monitoring components, terminate them or add new ones, by interacting a main application which is always alive. Such an application should be able to understand these needs, activate or deactivate necessary monitoring elements during runtime. When these characteristics of the Remote Monitoring Systems are considered, it can be easily observed that DSPL Engineering methodologies and product building practices are well matched with such systems. “Understanding monitoring needs and matching these needs to existing software components” can be expressed by the “context awareness and dynamic binding” mechanisms. Besides that, “loading or unloading monitoring components during runtime without restarting the application” can be expressed by the “dynamic reconfiguration” mechanism of DSPL Engineering. With this motivation, Remote Monitoring of Computer Systems was considered and it was decided to build a DSPL to establish infrastructure for remote monitoring systems.

This study initially presents a Feature Model for Remote Monitoring Systems. C&V analysis which is the building block for the SPLE has been carried out for the remote monitoring of computer systems. The study also provides a detailed feature dictionary together with cardinality information, feature attributes, constraints and extra functional features to constitute well-functioning DSPL for RMM systems.

To satisfy the altering monitoring needs for such systems, followings are offered as fundamentals of the framework presented on this study:

- Reusable dynamic feature binding mechanism for monitoring needs (With the help of IFeatureActivator and ConfigurationValidator classes)
- Reusable dynamic plugin loading mechanism by extending Qt Plugin Loading mechanisms at runtime (With the help of IPluginLoader and IPluginFactory classes)
- Reusable monitor factory plugin interface (With the help of PluginFactoryInterface, MonitorPluginInterface and MonitorWorker interfaces)

The case study presented on Chapter 5 demonstrates a product named “minimal RMM Agent”. This agent has been built with the use of the DSPL offered in this study and tested in the demo environment of COMODO, a company operating in the field of ITSM. The case study “Minimal RMM Agent” shows that this study keeps its promise to create a DSPL for Remote Monitoring systems.

This study contributes to Dynamic Software Product Line Engineering field with a living DSPL example. By managing variabilities on ITSM Remote Monitoring domain explicitly, a DSPL for IT Service Management Products has been developed. This product line has enabled COMODO to complete their products more economically and quickly. A functioning DSPL instance is a promising work for this

domain considering that although this subject is extensively studied in the academic world, it is difficult to come across published practices in the business world.

The framework mainly focuses on the functional requirements of remote monitoring system. Non-functional requirements of the system will be studied as a future work.

Dynamic feature attribute definition as an extension for Extended Feature Modeling technique is given in this study but as it is stated on the Feature Modeling chapter, handling such attributes within the framework is quite challenging therefore it is left as future work also.

ITSM practices often use the concepts of monitoring and managing together. During the monitoring of the system, the user can define extra operations according to the results of monitors and these operations are counted as “management”. For example, if an unknown process is activated on the computer and this state change is caught by the system, MSP can run a script to kill this process. Such operations like automatic script running, creating an SD ticket, automatic email sending operations are management actions and framework does not provide software assets and mechanism to handle such actions. They are planned to be considered and implemented in the future.

REFERENCES

Arraj, V. (2010). ITIL®: The Basics. *Buckinghampshire, UK*.

Bagheri, E., Di Noia, T., Ragone, A., & Gasevic, D. (2010). Configuring Software Product Line Feature Models Based on Stakeholders' Soft and Hard Requirements. *International Conference on Software Product Lines* (pp. 16-31). Jeju Island, South Korea: Springer.

Benavides, D., Trinidad, P., & Ruiz-Cortes, A. (2005). Automated reasoning on feature models. *International Conference on Advanced Information Systems Engineering* (pp. 491-503). Heidelberg: Springer.

Blair, G., Bencomo, N., & B. France, R. (2009, October). Models@Runtime. *IEEE Computer Society vol. 42, no. 10*, pp. 22-27.

Böckle, G., Muñoz, J. B., Knaube, P., Krueger, C., do Prado Leit, J. S., van der Linden, F., . . . Weiss, D. (2002, August). Adopting and Institutionalizing a Product Line Culture. *International Conference on Software Product Lines* (pp. 49-59). Berlin,Heidelberg: Springer.

Böckle, G., Pohl, K., & van der Linden, F. (2005). A Framework for Software Product Line Engineering. In *Software Product Line Engineering* (pp. 19-38). Heidelberg, Berlin: Springer.

Capilla, R., Bosch, J., & Kyo-Chul, K. (2013). Variability Modeling. In C. Rafael, J. Bosch, & K. Kyo-Chul, *Systems and Software Variability Management* (p. 32). Springer.

Capilla, R., Trinidad, P., Bosch, J., Ruiz-Cort's, A., & Hinchey, M. (2014). An Overview of Dynamic Software Product Line Architectures and Techniques: Observations From Research and Industry. *Journal of Systems and Software*, 3-23.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2009). Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer, 42(10)*, 37-43.

Cetina, C., Pelechano, V., Trinidad, P., & Cort's, A. R. (2008). An Architectural Discussion on DSPL. *In SPLC (2)*, (pp. 59-68).

Coplien, J., Hoffman, D., & Weiss, D. (1998). Commonality and Variability in Software Engineering. *IEEE Software, 15(6)*, 37-45.

Czarnecki, k., & Eisenecker, U. (2000). *Generative programming: methods, tools, and applications.* Addison-Wesley.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2005a). Formalizing Cardinality-based Feature Models and their Specialization. *Software process: Improvement and practice, 10(1)*, pp. 7-29.

Czarnecki, K., Helsen, S., & Eisenecker, U. (2005b). Staged Configuration through Specialization and Multilevel Configuration of Feature Models. *Software Process: Improvement and Practice, 10(2)*, pp. 143-169.

Dhungana, D., Rabiser, R., & Grünbacher, P. (2007). Decision-Oriented Modeling of Product Line Architectures. *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)* (pp. 22-22). IEEE.

Griss, M., Favaro, J., & d'Alessandro, M. (1998). Integrating Feature Modeling with the RSEB. *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)* (pp. 76-85). IEEE.

Hallsteinsen, S., Stav, E., Solberg, A., & Floch, J. (2006). Using Product Line Techniques to Build Adaptive Systems. *10th International Software Product Line Conference (SPLC'06)* (pp. 10 pp.-150). Baltimore, MD: IEEE.

Hinz, D., & Gewald, H. (2006). The Next Wave in IT Infrastructure Risk Management: A Causal Modeling Approach with Bayesian Belief Networks.

In M. Khosrow-Pour, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2.* Idea Group Publishing.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., & Peterson, A. S. (1990). *Feature-oriented domain analysis (FODA) feasibility study (No. CMU/SEI-90-TR-21).* Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Annals of Software Engineering 19(4) 5.1*, 143.

Kang, K. C., Lee, J., & Donohoe, P. (2002). Feature Oriented Product Line Engineering. *IEEE Software, 19(4)*, 58-65.

Kumbakara, N. (2008). Managed IT services: the role of IT standards. *Information Management & Computer Security,16(4)*, 336-359.

Lauenroth, K., & Pohl, K. (2005). Principles of Variability. In K. Pohl, G. Böckle, & F. van der Linden, *Software Product Line Engineering Foundations, Principles and Techniques* (pp. 58-88). Berlin, Heidelberg: Springer.

Metzger, A., & Pohl, K. (2007). Variability Management in Software Product Line Engineering. *Companion to the proceedings of the 29th International Conference on Software Engineering* (pp. 186-187). IEEE Computer Society.

Metzger, A., & Pohl, K. (2014). Software product line engineering and variability management: achievements and challenges. *Proceedings of the on Future of Software Engineering* (pp. 70-84). ACM.

Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., & Solberg, A. (2009). Models@ Run.time to Support Dynamic Adaptation. *Computer 42(10), no. 10*, 44-51.

Ridley, G., Young, J., & Carroll, P. (2004). COBIT and its Utilization: A framework from the literature. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on* (pp. 8-pp). IEEE.

Sahibudin, S., Sharifi, M., & Ayat, M. (2008). Combining ITIL, COBIT and ISO/IEC 27002 in Order to Design a Comprehensive IT Framework in Organizations. *Second Asia International Conference on Modelling & Simulation (AMS)* (pp. 749-753). IEEE.

Schmid, K., & John, I. (2004). A customizable approach to full lifecycle variability. In *Science of Computer Programming 53* (pp. 259-284).

SearchITChannel. (2015, July). *What is RMM software (remote monitoring and management software).* Retrieved from http://searchitchannel.techtarget.com/definition/RMM-software-remote-monitoring-and-management-software

Sena, J., & Obispo, S. L. (2006). Outsourcing, Insourcing IT-Related Business: The Impact on the Organization. In M. Khosrow-Pour, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2.* Idea Group Publishing.

The Qt Company. (2018, December 17). *QPluginLoader Class*. Retrieved from Qt Documentation: https://doc.qt.io/qt-5.6/qpluginloader.html

The Qt Company. (2018, December 17). *Signals & Slots*. Retrieved from Qt Documentation: http://doc.qt.io/qt-5/signalsandslots.html

The Qt Company. (2018, December 17). *The Low-Level API: Extending Qt Applications*. Retrieved from How to Create Qt Plugins: https://doc.qt.io/qt-5.6/plugins-howto.html

Von Solms, B. (2005). Information Security governance: COBIT or ISO 17799 or both? *Computers & Security 24(2)*, 99-104.