We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

2

An Approach for Representing Domain Requirements and Domain Architecture in Software Product Line

Shahliza Abd Halim, Dayang N. A. Jawawi,
Noraini Ibrahim and Safaai Deris
Software Engineering Department,
Universiti Teknologi Malaysia, Skudai,
Malaysia

1. Introduction

Software Product Line (SPL) core assets development is an effective approach in software reuse in which core assets can be shared among the members of the product line with an explicit treatment of variability. Among the artefacts of core asset are architecture, reusable software components, domain models, requirements statements, documentation and specifications, performance models, schedules, budgets, test plans, test cases, work plans, and process descriptions.Variability in its own right is the central concept in SPL which is not being catered by conventional method of reuse. Consequently, it is important for variability to be identified and to be represented early at requirements phase. The importance of identifying requirements variability earlier at requirements level is also known as systematic reuse by researchers (Frakes and Isoda 1994; Muthig 2002). Variability at requirements levels also initiates the existence of the variability at architecture thus further highlight the inadequacy of considering variability solely at architectural level. Therefore, considering on variability at architecture and its implementation level is not enough where the understanding of variability at the requirements level is also required (Yu, Akhihebbal et al. 1998; Moon 2005; Kircher, Schwanninger et al. 2006).

Nonetheless, there are challenges on relating variability at both abstraction levels where mapping of user requirements with the core assets for the adaptation process and derivation of core assets based on user requirements is a complex task (Matinlassi 2004; Dhungana 2006). This task is made difficult due to the dependencies among variants in architecture in order to fulfil a single customer's requirements (Bachmann and Bass 2001; Chastek 2001; Thiel and Hein 2002). Furthermore, the variability information assembled within the requirements phase should be able to support the following phase, the architecture design (Brown, Gawley et al. 2006). Consequently, the relationships between both abstraction levels are not always apparent especially between high level requirements artifacts and more specific and formal artifacts of architecture such as Architecture Description Language (ADL)(Medvidovic, Grünbacher et al. 2003). In addition, relating between requirements to

architecture also requires design decision to be explicitly represented (Bosch 2004; Avgeriou, Kruchten et al. 2007). Avgeriou et al. further highlight the importance of design decision accompanying the architecture development. Without the first class representation of explicit knowledge and rationale as in design decision, it leads to knowledge vaporization phenomena as described by Bosh. It is further suggested by him software architecture should also consider composition of domain models, usage scenarios, feature and other elements, which support architectural design decision.

In order to address the issues in relating between different abstraction levels, researchers proposed different views represented by different models with defined mappings between the models. The usage of multiple modeling and mappings are done by (Savolainen, Vehkomäki et al. 2002; Medvidovic, Grünbacher et al. 2003; Lee and Kang 2004; Savolainen, Oliver et al. 2005; Dhungana 2006; Sochos, Riebisch et al. 2006; Zhang, Mei et al. 2006; Zhu, Yuqin et al. 2007; Gomaa and Shin 2008; Bragança and Machado 2009; Lin, Ye et al. 2010). Among the approaches, Gomaa and Shin has the most comprehensive models used in their mappings (Gomaa and Shin 2008). Nevertheless, they only considers mapping at requirements to analysis model and do not involve mapping at architectural levels. There are also approaches which only concentrate on the rule and also the formal representation of the mapping without using any explicit models to represent the different abstraction levels (Savolainen, Oliver et al. 2005; Zhu, Yuqin et al. 2007). Furthermore, the mapping to architecture is generally referred as architectural assets and no specific elements mentioned at the architectural level. Another approach is by feature-driven mapping which is among the most accepted approaches so far by researchers (Lee and Kang 2004; Dhungana 2006; Sochos, Riebisch et al. 2006; Zhang, Mei et al. 2006; Lin, Ye et al. 2010). Nevertheless, these approaches seldom have an explicit representation of design decisions in order to records decisions that architects made while designing the domain architecture.

Therefore, even though the above-mentioned approaches for transitioning requirements models to architecture levels have proposed techniques to overcome majority of the issues mentioned earlier, nevertheless there are still room for improvement in the focus on of both functional and non-functional requirements which are essential elements for architecture development and also on the transition process itself which cannot be fully automated thus highlighting the importance of design decision in bridging between requirements and architectural level (Paech, Dutoit et al. 2002; Kaindl and Falb 2008; Turban, Kucera et al. 2009). However, we only elaborate on design decision at the conceptual framework only and both requirements and architecture level representation will be the center of attention instead in this chapter.

The layout of this chapter is as follows: Section 2 discusses the conceptual framework and process governing the representation of domain requirements and domain architecture. In Section 3, metamodels representing domain requirements level will be discussed. Discussion on the metamodels representing domain architecture level is described in Section 4. Section 5 illustrates the usage of the representation in Autonomous Mobile Robot Product Line case study. Section 6 discusses on the evaluation of the proposed notation. Lastly, Section 7 concludes this chapter.

2. Conceptual framework for bridging between domain requirements and domain architecture

In order to address the issues of integrating functional, non-functional, architecture and design decisions in relating between the two abstractions levels, we argue that SPL architecture design method should incorporate multiple model approach in order to relate the requirements elements to architectural elements. Multiple model approach can provide different views of the system for different stakeholders. Furthermore, in order to have a clear identification and representation for requirements to enable it to be of importance at the latter development phase, we investigate the knowledge suitable to be incorporated in domain requirements profile and also at domain requirements profile for the purpose of assisting the domain architecture representation development. Therefore, the research question to be answered in this book chapter is "*What are the representations suitable for representing core assets at domain requirements level*?" and "*What are the representations suitable for representing core assets at domain architectural level?*"

The conceptual framework for relating from requirements to architecture follows the framework proposed by Garlan, Capilla and Babar (Garlan 2000; Capilla and Ali Babar 2008) as shown in Figure 1. Garlan proposed an architecture representation by incorporating ADL in object oriented modeling UML.Whereas, Capilla and Babar proposed on three different elements should be incorporated in a decision mode the product constraints, variability and binding.

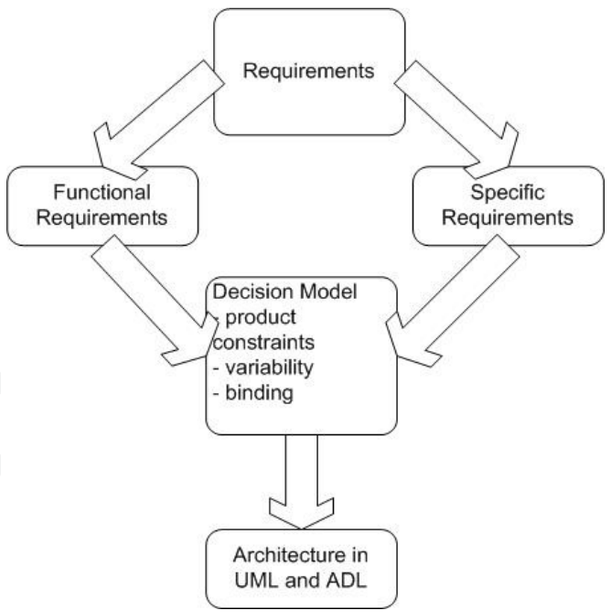

Fig. 1. Conceptual Framework for relating between requirements model to architecture

In order to represent the requirements and architectural elements representation, we use the extension provided in SysML profile (SysML 2006). SysML has extended UML 2.0 with specialized support for requirements engineering and traceability elements between requirements model and other models elements thus making it a perfect candidate in

support of the variability extension. Furthermore, traceability support in SysML can contribute to the possibility of mapping between requirements and architectural level being done in the language. Section 3 and Section 4 will elaborate more on the treatment and mapping for each model.

Based on Figure 2, except for **Requirements Context** which is a matrix table to analyse requirements commonality and variability, **Use Case model, Parametric model** and **Feature Model** are multiple models for representing Domain Requirements. Parametric diagram is a new diagram extension in SysML to represent constrain on the property or behavior of a system (Friedenthal, Moore et al. 2008; Holt and Perry 2008). The diagram is used for representing system equations that can constrain the properties of a block (Friedenthal, Moore et al. 2008). Though the model is usually used to represent constraint in terms of mathematical equations it has the potential to be extended to represent general rule or to apply for requirements validation and verification. Figure 2 shows the association between the domain requirements model and the domain architectural model where design decision is the connection which link between the models. Decision model is where the functional and non functional constraint is being specified. There are also two types of mapping based on the figure, horizontal mapping refers to mapping between models at the same level of abstraction, in this case at domain requirements level (between use case, feature and parametric model). Vertical mapping refers to the mapping between different levels of abstractions, between domain requirements model and domain architecture model. Decision model will be the intermediate model between both abstraction levels.

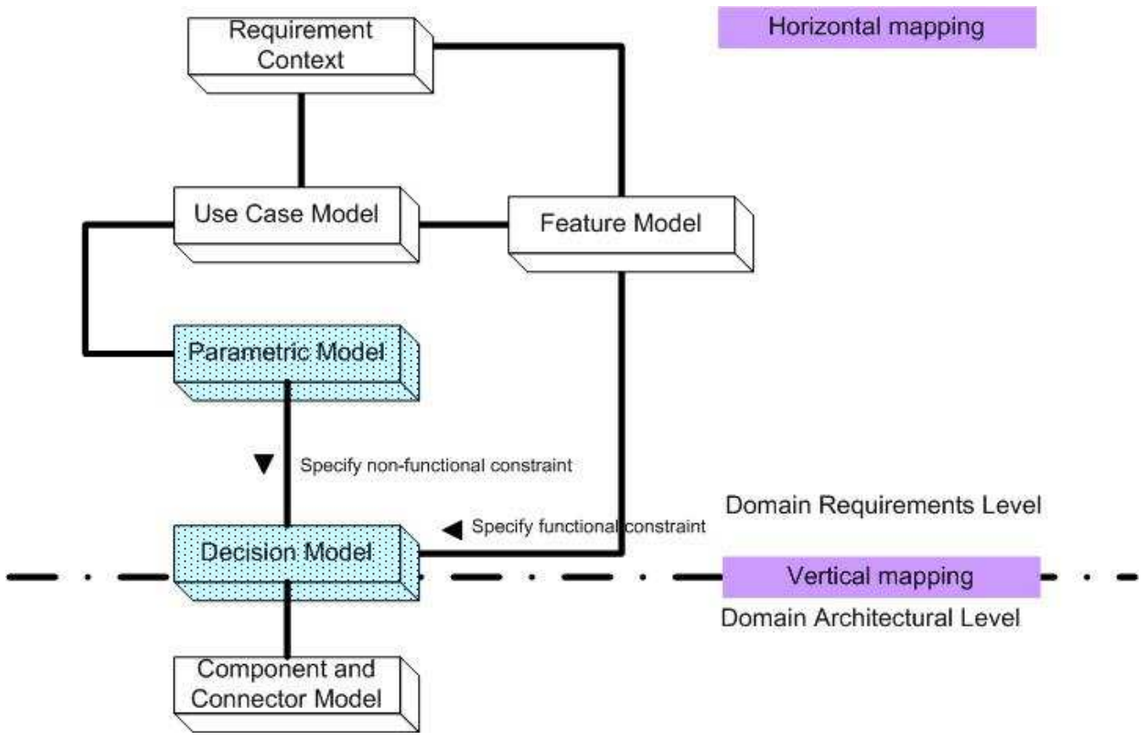

Fig. 2. Model for associating domain requirements to domain architecture with decision model

The lack of fundamental process models and guidelines for the transition between the two abstraction levels, further hinders the systematic task in developing the architecture. In order to have a clear process for relating between requirements to architecture in SPL, we look into available PLA design methods itself on their support for an explicit functional and non functional requirements and its transition for architecture design. Existing method for Product Line architecture design based on comparison by Matinlassi (Matinlassi 2004) has evaluated COPA, FAST, FORM, KobrA and Quality Driven Architecture Design and Analysis (QADA). From the evaluation, QADA method has consideration on quality attributes requirements. We have also reviewed books on SPL such as by (Gomaa 2005) concentrating on Product Line UML based Software Engineering (PLUS) method and by Bosch proposing Functionality based Architecture Design (FAD) method (Bosch 2000).

From the reviews there are only two architecture design methods that focus on functional and non-functional requirements, QADA and FAD. However, we concentrate on FAD as a process in our research as it provides clear description of its Product line Architecture process. Though FAD has a concentration in functional and non-functional requirements, yet it still does not show explicitly what are the techniques or methods involve for the process at the requirements level. Based on Figure 3, we will add suitable methods for each part of the processes in FAD (i.e. requirements specification, software architecture, design decision, derivation and mapping and lastly the evaluation or assessment done to the architecture).

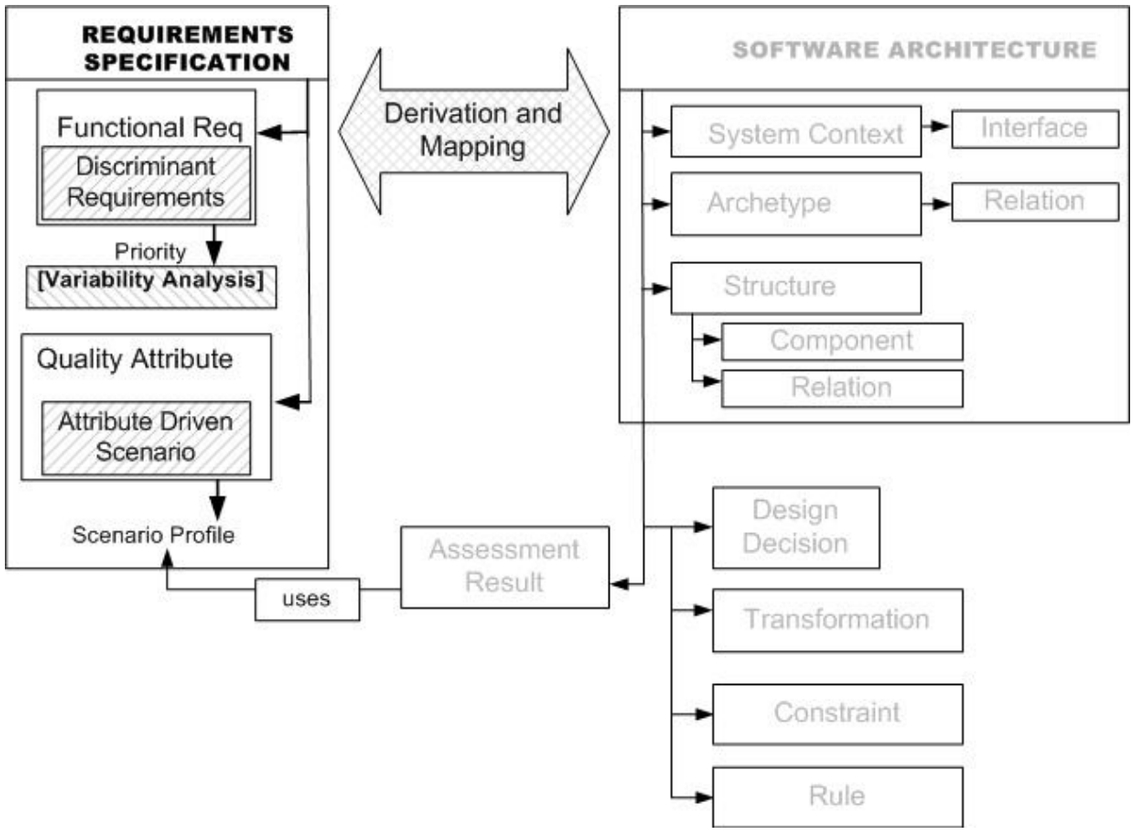

Fig. 3. Process for associating domain requirements to domain architecture adapted with enhancement from (Bosch 2000)

3. Metamodel for representing core assets at requirements level

Due to the unstructured nature of requirements, there are several approaches which combined different strategies in order to represent artifacts in requirements analysis for SPL. A systematic review has reported the high usage of textual and features artifacts in domain analysis followed by use cases and goal based methods and others (Mahvish and Tony 2009). Albeit the popular usage of feature model by various researches in SPL, it does not properly represent variability information (Bühne, Lauenroth et al. 2004; Moon 2005). Among the variability information that could not be supported by feature model are such as proper decision on choosing features as either common or optional, identification of variation points and also variation point type (Moon 2005) and required behavioral information in its representation (Brown, Gawley et al. 2006). Goal based strategy also has been reportedly having its own problem of implementation such as the abstractness of its concept has leads to the problem in finding the right goal (Aurum and Wohlin 2005). Thus, in our research, we concentrate on determining objectively the common and variable feature based on the analysis on existing similar applications. To achieve the objectivity, commonality and variability matrix is used in order to identify which are the common and optional requirements (Mikyeong, Keunhyuk et al. 2005; Halim, Jawawi et al. 2009). In order to complement the use of feature model, use case model is chosen as it enables the representation of text based system behavior (Armour, Miller et al. 2001).

a. Functional mapping

For functional mapping, the feature model is used for representing functional requirements while use case represents the behavioral specification of the requirements. Use case model have two extensions to its metamodel where use case documentations have been added with extra parameters for describing quality attributes. The extensions are shaded in grey as shown in Figure 4. Another extension is on use case types to identify priority and reuse property of the use case. For example if the priority is high the use case is a common and will be reused by all the application in the product line.

FODA is commonly used as feature model by researcher, however in this research, feature metamodel MRAM/TRAM is used as it has already proposed an extension of SysML profile (Mannion and Kaindl 2008). The metamodel contains discriminants which are features that differentiate one system from another. Discriminant and its associated pattern comprise of single adaptor, multiple adaptor and option. The stereotypes <<MA>> represent multiple adaptors, where at least one of the requirements can be chosen, while <<SA>> represent Single Adaptor variability where only one requirement can be chosen from the variants. Furthermore, MRAM/TRAM paired parameters and discriminant for modeling qualitative and quantitative variability. According to (Magnus, Jurgen et al. 2009), discriminant provide a decision model for composing product specification from product line requirements documentation. Figure 4 shows the mapping between use case model and feature model.

b. Non functional mapping

For representing non functional requirements, architecture scenario is used (Clements, Bachmann et al. 2003; Liming, Babar et al. 2004; Oquendo, Warboys et al. 2004; Bachmann, Bass et al. 2005). With the use of architecture scenario, the non functional requirements can be represented with more attribute instead of using just a general description or only using

one word such as Performance, Modifiability or Security. The architecture scenario comprise of six elements which we will further refer as non functional parameters: stimulus; source of stimulus; environment; artifact, response; response measure and expected response .(Bass, Clements et al. 2003). Previously, architecture scenario has been proposed as design decisions and non functional requirements by (Zhu and Gorton 2007). However, our practice of using architecture scenario is in parametric diagram where it explicitly shows the non functional parameters of the architectural scenario. To enable non functional parameters to be defined in parametric diagram, it has to be defined prior to its usage. The metamodel of the parametric diagram is based on Holt and Perry (Holt and Perry 2008).

c. Horizontal mapping

The functional mapping between use case and feature metamodel is referred as in Figure 4 and also (non-functional mapping) between use case and parametric metamodel in Figure 5 as horizontal mapping. Hence, horizontal mapping is between models at the same level of abstraction, in this case at domain requirements level. In use case metamodel, we refer Extension Point metaclass as a variation point in use case model and also Discriminant metaclass as a variation point in Feature model. Thus, based on Figure 4, we have defined the mapping between variation points at use case model with variation point at feature model.

For the mapping between use case and parametric metamodel in Figure 5, the mapping is more superficial due to the nature of non – functional requirements which not usually exist in each use case. Furthermore, non-functional requirements also known to have an impact to one whole application and again there is no specific use case that can show this type of information. Thus, we will dwell further into this matter as our future research.

4. Metamodel for representing core assets at architectural level

UML has been used as an architecture modeling language and also a de facto modeling language used in the industry, even so there are arguments concerning its modeling notations inadequacy for representing architecture (Medvidovic, Rosenblum et al. 2002; Medvidovic, Dashofy et al. 2007). Another paradigm, which has a consistent, complete and correct architecture description for representing architecture is by using Architecture Description Language (ADL) (Taylor, Medvidovic et al. 2009).

Integrating both languages, ADL and UML can be considered as having a synergistic relationship where the combination enables a precise and explicit architecture description and at the same time having a wider usage among UML users in commercial tool. xADL is chosen due to its specialized schema targeted for product line architecture description (Dashofy, Hoek et al. 2005) while SysML is chosen due to its first class consideration for requirements modeling and also its traceability elements between requirements model and other models elements.

a. Mapping of SysML to xADL

The metamodel of the xADL and SysML integration have been proposed in (Halim, Jawawi et al. 2009). We have divided the profile into three sections, the metaclass section which consists of UML classes reused in SysML known as UML4SysML. The architectural construct section which shows the extension of stereotype classes and the variability construct section which shows the extension of stereotype to represent variability.

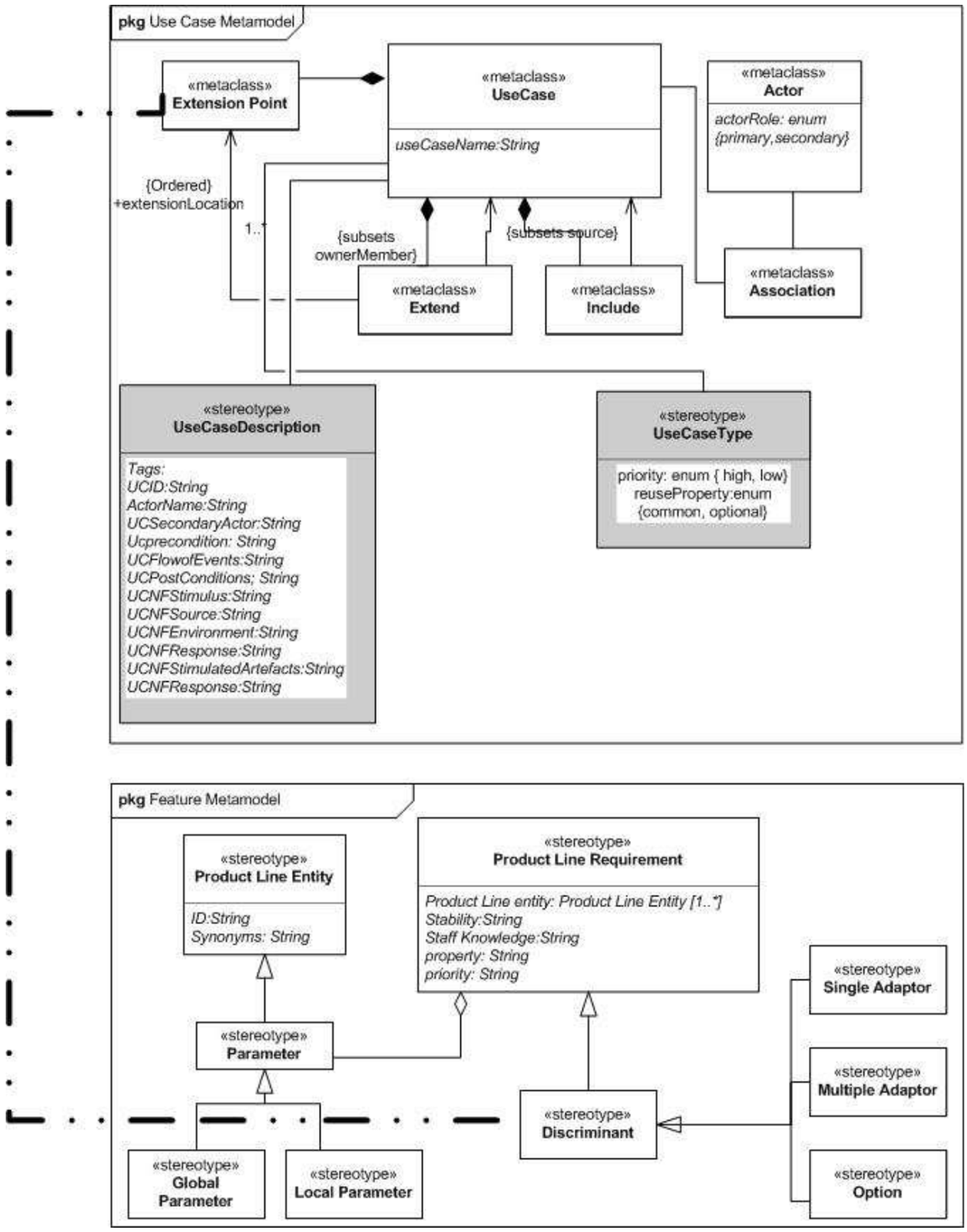

Fig. 4. Horizontal Mapping between use case and feature model

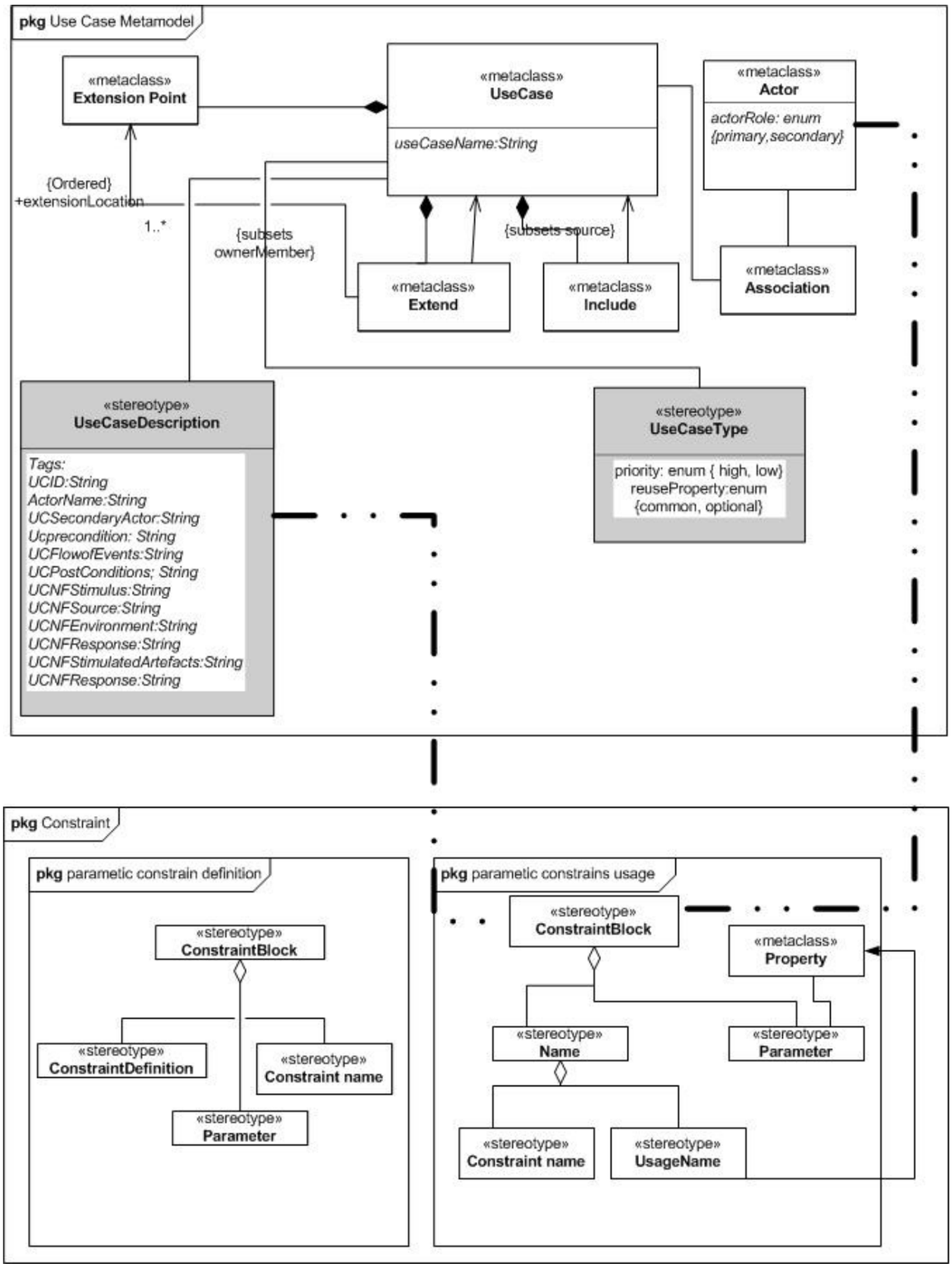

Fig. 5. Horizontal Mapping between use case and parametric model

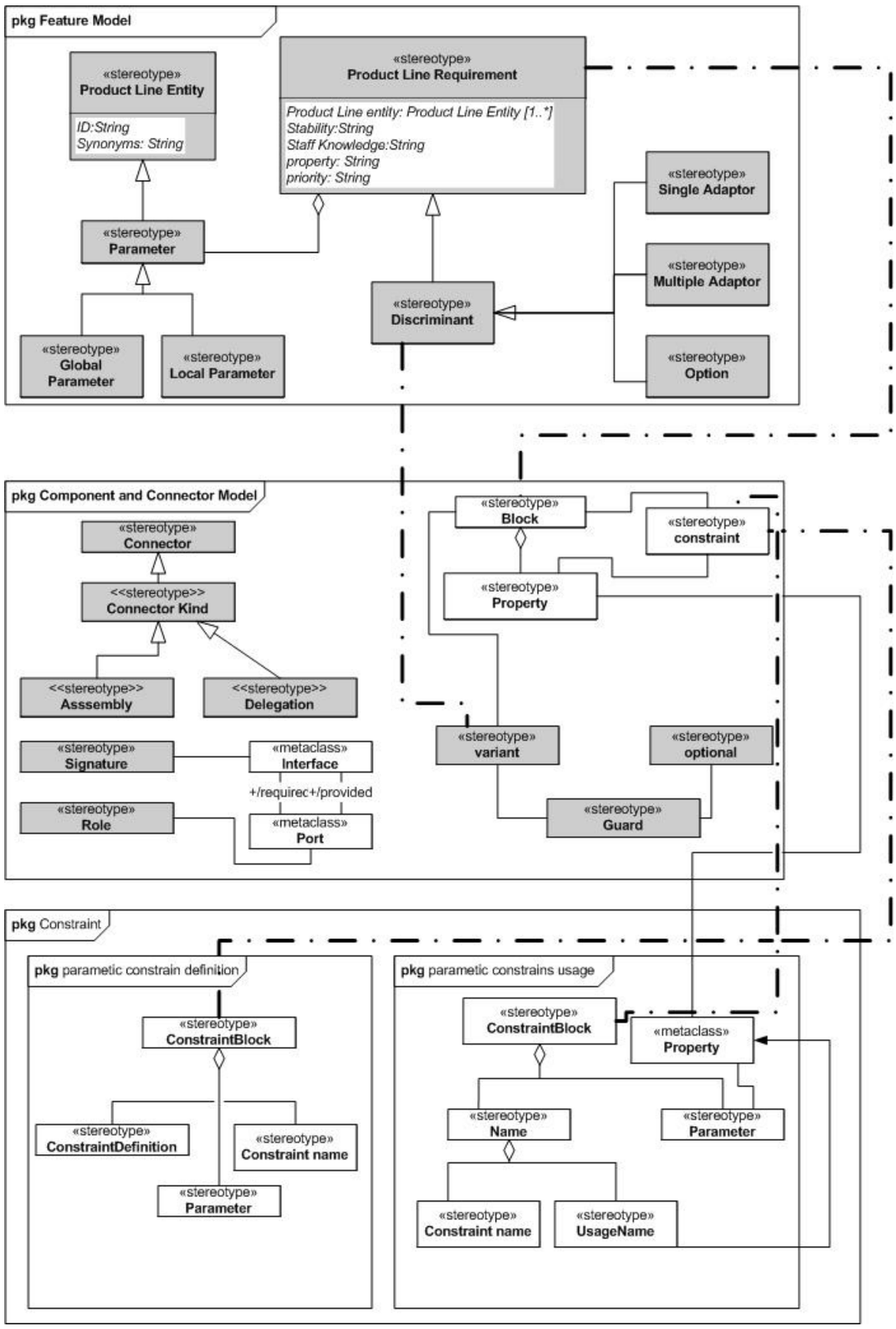

Fig. 6. Vertical Mapping between Domain Requirements to Domain Architecture

b. Vertical Mapping

Referring back to Figure 2, there should be a vertical mapping between domain requirements model (feature and parametric model) with the domain architecture model. However, the mapping involved decision model as an intermediate layers between the two abstraction levels. Due to the insufficient research result for decision model, the vertical mapping is done without considering decision model. Though the decision model does not exist, the vertical mapping shown in Figure 6 can be a future reference for capturing traceability information for the decision model.

The vertical mapping between the Feature metamodel to the component and connector metamodel is basically between the Discriminant metaclass to the Variant metaclass in Component and Connector metamodel. The mapping between the parametric diagram and the component and connector architecture is based on the constraint in the Block metaclass which can be matched to the *ConstraintBlock* metaclass in the Parametric metamodel.

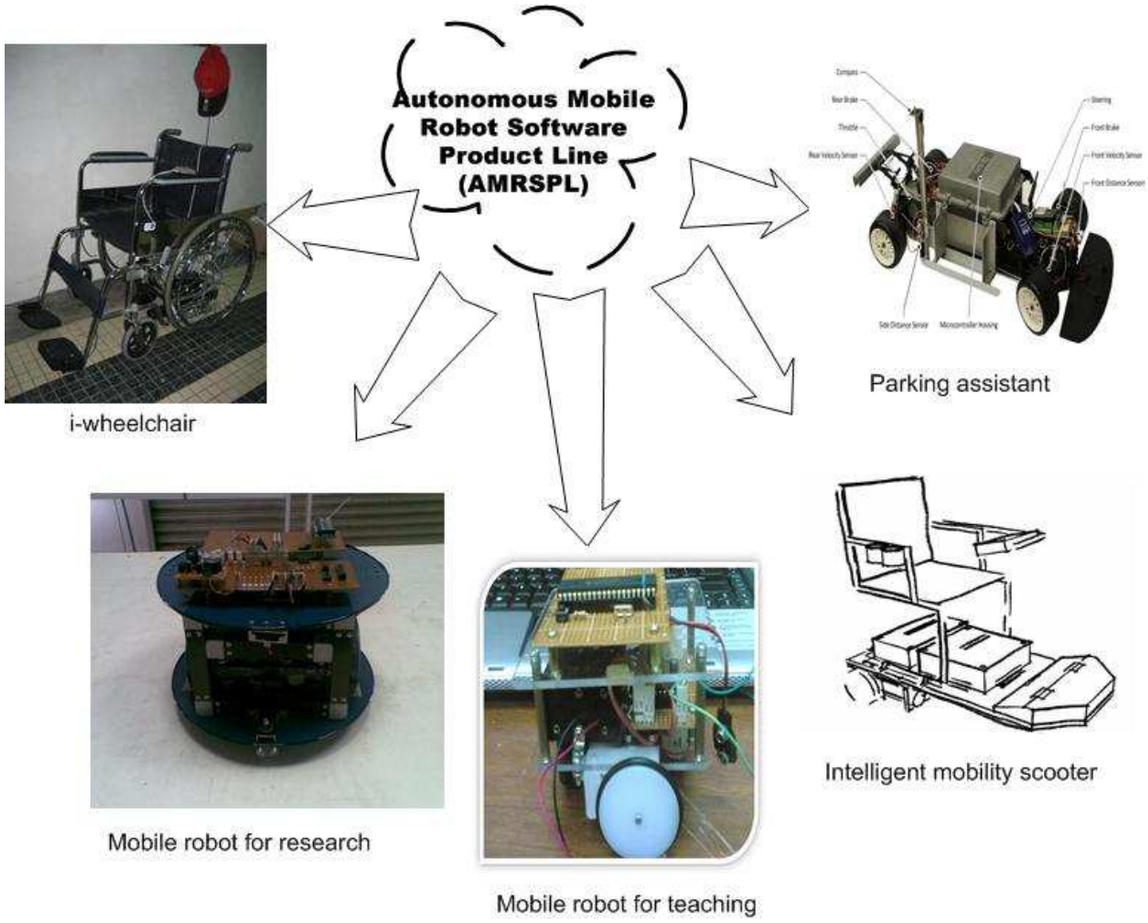

Fig. 7. AMR Product Line (AMRPL)

5. Case study of autonomous mobile robot software product line

In order to validate the applicability of the extended modelling in SysML, the extended model was applied to product line of Autonomous Mobile Robots (AMR). The product line consists of five different but similar applications of AMR. Four of the AMR are AMR for research, AMR for teaching, i-wheelchair and intelligent scooter based on the research collaboration done at Embedded Real Time and Software Engineering Research Lab (ERetSEL), Universiti Teknologi Malaysia. The fifth AMR is the parking assistant based on the work of Polzer, Kowalewski and Botterweck (Polzer, Kowalewski et al. 2009). The AMRPL is as shown in Figure 7.

In order to identify the commonality and variability of the AMRPL requirements, approach by Abd Halim, Jawawi and Safaai (Halim, Jawawi et al. 2009) is used. However, in order to simplify this paper, the common and variable function is represented in use case diagram as shown in Figure 8.

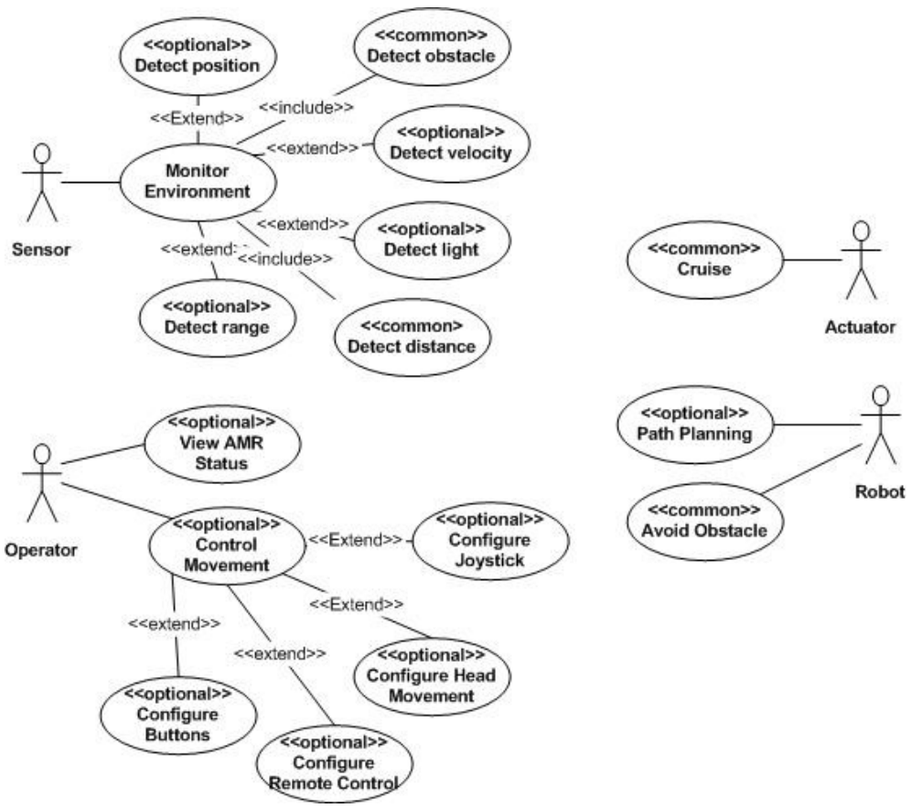

Fig. 8. AMRPL Use Case

Figure 9 shows all the three models based on their corresponding metamodel in Figure 6. Feature model represents the functional requirements in the form of SysML requirements model. Feature model in Figure 9 shows only partial requirements for AMRPL. The stereotypes <<MA>> represent multiple adaptors, where at least one of the requirements can be chosen, while <<SA>> represent Single Adaptor variability where only one requirement can be chosen from the variants. The ellipse shape for variants in Motor

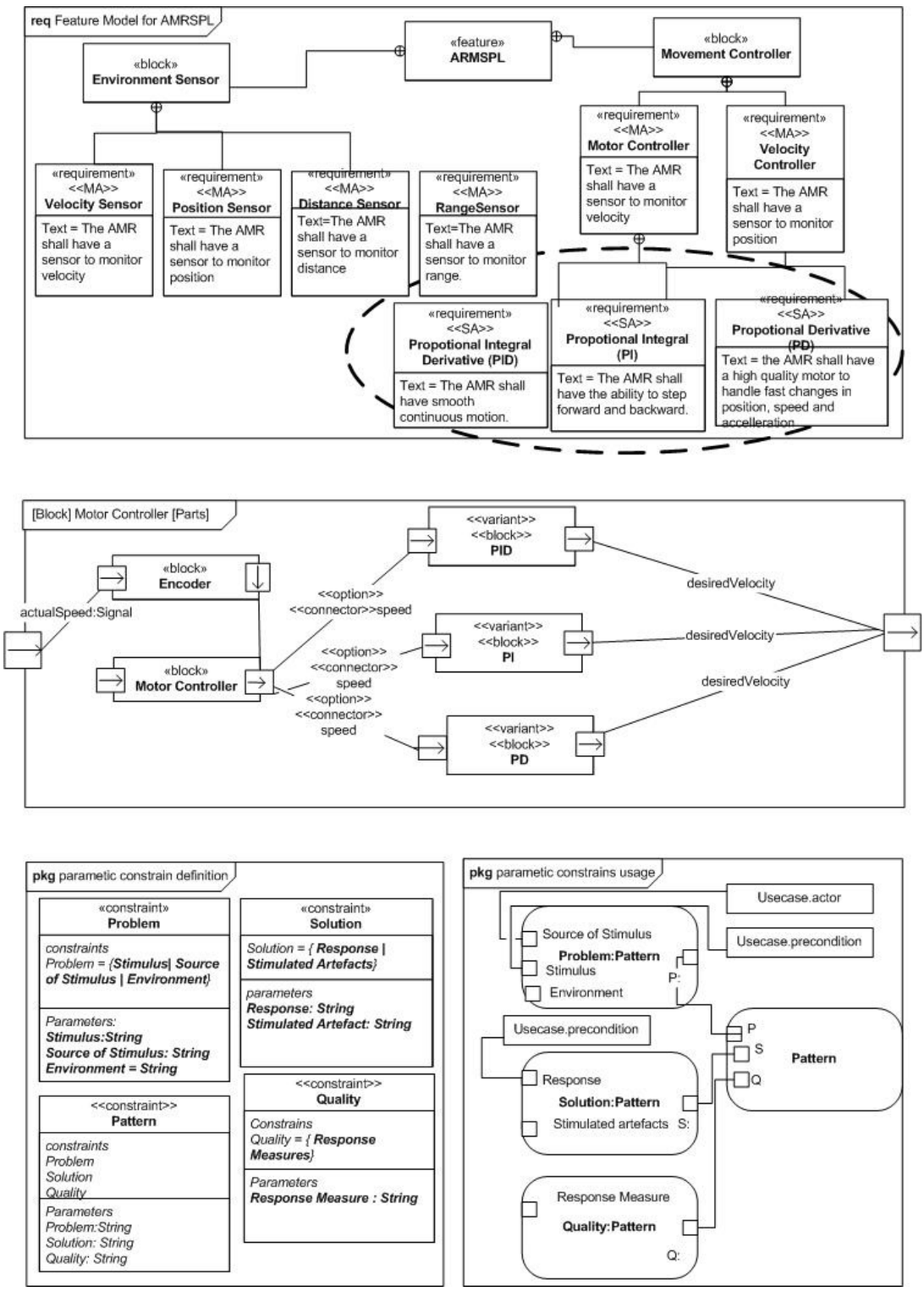

Fig. 9. Vertical mapping betweeen Feature Model, Parametric Model and Component Connector Model of AMRPL

Controller such as PID, PI and PD has been elaborated in the same name as in component and connector model. Non functional requirements are shown in parametric model in Figure 9. The parametric model is divided into definition and usage constraint. Parametric model in Figure 9 basically have defined four constraints. The parametric model represents the architecture scenario and how the scenario helps in identifying suitable patterns (Liming, Babar et al. 2004; Oquendo, Warboys et al. 2004). The pattern identified can then be used for later refinement of the initial architecture in the component and connector model.

6. Discussion

Based on the applicability of the proposed approach in modeling the domain requirements and domain architecture for AMRPL, we evaluate our proposed models and annotation in previous sections with suitable evaluation criteria. As far as our concern, there are two existing evaluation frameworks for evaluating variability modeling (Djebbi and Salinesi 2006; Sinnema and Deelstra 2007). The former proposed eleven criteria for comparing requirements variability modeling notations resulted from a brainstorming session with stakeholders. The evaluation framework is then compared to four feature-based notations FOPLE, FeatuRSEB, GP and FORE. The latter, concentrates on classifying variability modeling techniques based on two key characteristics. The key characteristics are representation for the variability itself and the tool accompanying the variability modeling (Sinnema and Deelstra 2007). Due to our variability modeling approach is based on profile extension, therefore the evaluation suitable for our approach is based on the extensions and the notations proposed in modeling the variability. Thus, we based our evaluation on the first evaluation framework proposed by Djebbi and Salinesi. The second evaluation framework by Sinnema and Deelstra is unsuitable for evaluating our approach as it rely heavily on the use of tool for the evaluation.

From the eleven criteria in the evaluation framework, we have classified the criteria into three classifications concerning our proposed notation. The three classifications are evaluation criteria fit for the notation, evaluation criteria for future extension of the notation and evaluation criteria which not covered by the notation. Evaluation criteria that fall into the first classification are readability of the notation, simplicity and expressiveness of the notation, explicit variability types of the notation, specification for variation point property, unified modeling of the notation and standardize notation. Second classifications, the future extension for the notation consist of criteria such as dependencies representation between the variable part of the product line, scalability of the notation and also the tool which support the proposed notation. The third classification is for the criteria which are not considered in our approach. Among the criteria are the evolution support of the product line and the adaptability of the notation towards other companies. Hence, we evaluate our notation based on the first classification only and the second and third classification will not be elaborated as it is either not being implemented yet or not related to our proposed notation.

Notations readability can be achieved by clear and minimal representation. The case study showed in previous section confirms visualization of variability at requirements and architectural level by using stereotypes. These elements thus demonstrate the clear representation of the notation. However, this notation has redundancy on information representing functional and non-functional requirements such as the information from use case to parametric diagram. This redundancy affect the minimality of the notation. For

simplicity and expressiveness criteria, the construct proposed in the metamodel can be considered as sufficient to represent variability at requirements and architectural level. Though there are multiple models involve in representing domain requirements and domain architecture, however the number of entities in the metamodel are higher than the number of its relations thus highlight the simplicity of the notation. Expressiveness criteria have possibility to be achieved as the notation is based on extension from UML constructs therefore it can be understood by the user without much explanation. However, new model such as parametric diagram will have a significant effort for comprehension.

Evaluation criteria fit for the notation	
Readability of the notation	Metamodel mapping and UML based notations help in defining the graphical means to visualize domain requirements and domain architecture. However, there is possible duplication on information representing functional and non-functional requirements.
Simplicity and expressiveness of the notation	Simplicity can be achieved with the minimal construct in the metamodel to show variability. SysML profile which is an extension of UML reflects the expressiveness criteria.
Explicit variability types of the notation	Variability types at requirements and architectural levels are considered.
Specification for variation point properties	The proposed notation has a clear representation for variability through the use of stereotypes in the notation.
Case tool support	SysML profile extension is conformed to standard UML hence can also be supported by existing UML tool.
Unified modeling of the notation	Notations at requirements level have traceable relationships to notations at architectural level.

Table 1. Evaluation criteria fit for the notation

The notation can fulfill the third evaluation criterion, by having an explicit variability type at requirements level such as in use case relationships of uses and extend, in feature diagram relationships as in single adaptor, multiple adaptor, and options and in component and connector relationships such as the use of variants and options. The fourth criterion is on the specification of the variation points. Though it's not being shown in the case study, we have proposed the specification of the variation points at the requirements level which can be referred in (Halim, Jawawi et al. 2009). Nevertheless, specification of variation points at architectural level is yet to be defined. The following evaluation criteria is on case tool support. With the use of SysML profile which extends from UML itself, the notation can be used in any tools which support UML. Nonetheless, a fully automated tool is still being designed in order to automatically manage the variability of the models. The last evaluation

criterion, unification in the proposed approach is achieved with the ability to transfer variability in models at both abstraction levels. At requirements level, variability information is transformed between use case, feature model and parametric diagram. The variabilities in both use case and feature model are then transferred to the component and connector model through decision model. Therefore, from the proposed mapping from each of the metamodels representing the use case, feature model, parametric model and component and connector model, an initial unified modeling of variability can be achieved.

Table 1 summarizes the discussion related to the first classification. Based on Table 1, the evaluation is done on our proposed notation only and there is no comparison done to other existing methods.

7. Conclusion

An initial mapping between multiple models at requirements level to an architectural model has been paved. The applicability of the approach has been validated in AMRPL case study. The proposed notations and annotation used to model the AMRPL have also been evaluated using an evaluation framework (Djebbi and Salinesi 2006). From the evaluation there are several criteria have been fulfilled by the proposed notation among them are its readability, simplicity and expressiveness, explicit variability types, specification for the variability, unified modeling and tool support. Nonetheless the mapping have not yet consider design decision as an intermediate model for vertical mapping between domain requirements to domain architecture. The initial mapping contain basically a syntactic information of how it can possibly be done. The semantics and rules of the mapping is the future work of this research as these two elements are important for a more consistent approach of multiple model mapping. In this paper also, an initial use of parametric model to represent quality requirements has been shown. While it shows significant new way of using parametric diagram which previously known to only represent mathematical equations, nonetheless further refinement of how the model can be used to show the affect of non functional requirements at architectural levels is strongly needed. The proposed approach of using lightweight mechanism in representing the extension to map and represent the models at different levels of abstraction also need to be evaluated with a proper matrix to ensure its quality in representing PL architecture. Therefore, our future work is on refining the design decision model and how the rules at requirements and architecture level can be implemented in the design decision as a mapping between both abstraction levels.

8. Acknowledgement

This research is fully funded by the Research University Grant (RUG) from the Universiti Teknologi Malaysia (UTM) and Ministry of Higher Education (MOHE) under Cost Center No.Q.J130000.7128.03J23. Our profound appreciation also goes to ERetSEL lab members for their continuous support in the working of this paper.

9. References

Armour, F., G. Miller, et al. (2001). Advanced use case modeling: software systems, Addison-Wesley.

Aurum, A. and C. Wohlin (2005). Engineering and managing software requirements, Springer Verlag.

Avgeriou, P., P. Kruchten, et al. (2007). Sharing and Reusing Architectural Knowledge--Architecture, Rationale, and Design Intent, IEEE Computer Society.

Bachmann, F. and L. Bass (2001). Managing Variability in Software Architectures. Proceedings of the 2001 Symposium on Software reusability: Putting Software Reuse in Context Toronto, Ontario, Canada, ACM Press.

Bachmann, F., L. Bass, et al. (2005). Designing software architectures to achieve quality attribute requirements, IET.

Bass, L., P. Clements, et al. (2003). Software architecture in practice, Addison-Wesley Longman Publishing Co., Inc.

Berg, K., J. Bishop, et al. (2005). Tracing Software Product Line Variability: from Problem to Solution Space. Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, White River, South Africa South African Institute for Computer Scientists and Information Technologists.

Bosch, J. (2000). Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, Addison-Wesley.

Bosch, J. (2004). "Software architecture: The next step." Lecture notes in computer science: 194-199.

Bragança, A. and R. J. Machado (2009). "A model-driven approach for the derivation of architectural requirements of software product lines." Innovations in Systems and Software Engineering 5(1): 65-78.

Brown, T. J., R. Gawley, et al. (2006). Weaving behavior into feature models for embedded system families, Baltimore, MD, United states, Inst. of Elec. and Elec. Eng. Computer Society.

Brown, T. J., R. Gawley, et al. (2006). Weaving behavior into feature models for embedded system families.

Bühne, S., K. Lauenroth, et al. (2004). Why is it not Sufficient to Model Requirements Variability with Feature Models?

Capilla, R. and M. Ali Babar (2008). On the Role of Architectural Design Decisions in Software Product Line Engineering Software Architecture, Springer Berlin / Heidelberg. 5292: 241-255.

Chastek, G. (2001). Product Line Analysis: A Practical Introduction. Pittsburgh, Software Eng. Inst. (SEI), Carnegie Mellon Univ.

Clements, P., F. Bachmann, et al. (2003). Documenting software architectures: views and beyond, Addison-Wesley, Boston.

Dashofy, E. M., A. Hoek, et al. (2005). "A comprehensive approach for the development of modular software architecture description languages." ACM Transactions on Software Engineering and Methodology (TOSEM) 14(2): 199-245.

Dhungana, D. (2006). Integrated variability modeling of features and architecture in software product line engineering. 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06), Tokyo, Japan, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

Dhungana, D., R. Rabiser, et al. (2007). "Decision-oriented modeling of product line architectures."

Friedenthal, S., A. Moore, et al. (2008). A Practical Guide to SysML: Systems Model Language, Morgan Kaufmann.

Garlan, D. (2000). "Software architecture: a roadmap." Proceedings of the Conference on The Future of Software Engineering: 91-101.

Gomaa, H. (2005). Designing Software Product Lines with UML. From use cases to pattern-based software Architectures, Addison Wesley.

Gomaa, H. and M. E. Shin (2008). "Multiple-view modelling and meta-modelling of software product lines."

Halim, S. A., D. N. A. Jawawi, et al. (2009). Requirements Identification and Representation in Software Product Line. Asia Pacific Software Engineering Conference (APSEC'09), Pulau Pinang, Malaysia, IEEE.

Holt, J. and S. Perry (2008). SysML for systems engineering, Institution of Engineering & Technology (IET).

Kaindl, H. and J. Falb (2008). Can We Transform Requirements into Architecture? International Conference on Software Engineering Advances (ICSEA'08) IEEE Computer Society.

Kandé, M. M. and A. Strohmeier (2000). Towards a UML profile for software architecture descriptions, Springer-Verlag.

Kircher, M., C. Schwanninger, et al. (2006). Transitioning to a software product family approach - challenges and best practices. Software Product Line Conference, 2006 10th International.

Lee, K. and K. C. Kang (2004). "Feature dependency analysis for product line component design." Software Reuse: Methods, Techniques and Tools: 69-85.

Liming, Z., M. A. Babar, et al. (2004). Mining patterns to support software architecture evaluation. Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on.

Lin, Y., H. Ye, et al. (2010). An Approach for Modelling Software Product Line Architecture. International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, IEEE.

Magnus, E., B. Jurgen, et al. (2009). "Managing requirements specifications for product lines - An approach and industry case study." J. Syst. Softw. 82(3): 435-447.

Mahvish, K. and G. Tony (2009). "A systematic review of domain analysis solutions for product lines." J. Syst. Softw. 82(12): 1982-2003.

Mannion, M. and H. Kaindl (2008). "Using parameters and discriminants for product line requirements." Systems Engineering 11(1).

Matinlassi, M. (2004). Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. Proceedings of the International Conference on Software Engineering (ICSE'04), IEEE.

Medvidovic, N., E. M. Dashofy, et al. (2007). "Moving architectural description from under the technology lamppost." Information and Software Technology 49(1): 12-31.

Medvidovic, N., P. Grünbacher, et al. (2003). "Bridging models across the software lifecycle." Journal of Systems and Software 68(3): 199-215.

Medvidovic, N., D. S. Rosenblum, et al. (2002). "Modeling software architectures in the Unified Modeling Language." ACM Transactions on Software Engineering and Methodology (TOSEM) 11(1): 2-57.

Mikyeong, M., Y. Keunhyuk, et al. (2005). "An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line." Software Engineering, IEEE Transactions on 31(7): 551-569.

Moon, M., Yeom, K and Chae, HS (2005). "An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in Product Line." IEEE Transactions on Software Engineering 31(7): 551-569.

Oquendo, F., B. Warboys, et al. (2004). Distilling Scenarios from Patterns for Software Architecture Evaluation – A Position Paper. Software Architecture, Springer Berlin / Heidelberg. 3047: 225-229.

Paech, B., A. H. Dutoit, et al. (2002). Functional requirements, non-functional requirements, and architecture should not be separated - A position paper. REFSQ, Essen.

Polzer, A., S. Kowalewski, et al. (2009). Applying Software Product Line Techniques in Model-based Embedded Systems Engineering. MOMPES 2009, Vancouver, Canada.

Savolainen, J., I. Oliver, et al. (2005). Transitioning from product line requirements to product line architecture, Edinburgh, Scotland, United Kingdom, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States.

Savolainen, J., T. Vehkomäki, et al. (2002). "An Integrated Model for Requirements Structuring and Architecture Design." Proceedings of the Seventh Australian Workshop on Requirements Engineering, Melbourne.

Schmid, K. and I. John (2004). "A customizable approach to full lifecycle variability management." Science of Computer Programming 53(3): 259-284.

Sochos, P., M. Riebisch, et al. (2006). The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering and Computer Based Systems (ECBS'06).

Taylor, R. N., N. Medvidovic, et al. (2009). "Software Architecture: Foundations, Theory, and Practice."

Thiel, S. and A. Hein (2002). Systematic Integration of Variability into Product Line Architecture Design. Software Product Lines : Second International Conference, SPLC 2, San Diego, CA, USA, August 19-22, 2002. Proceedings: 67-102.

Turban, B., M. Kucera, et al. (2009). Bridging the Requirements to Design Traceability Gap Intelligent Technical Systems, Springer Netherlands. 38: 275-288.

Yu, C. C., A. L. Akhihebbal, et al. (1998). Handling Variant Requirements in Software Architectures for Product Families. Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families, Springer-Verlag London, UK.

Zhang, W., H. Mei, et al. (2006). "Feature-driven requirement dependency analysis and high-level software design." Requirements Engineering 11(3): 205-220.

Zhu, C., L. Yuqin, et al. (2007). "A Feature Oriented Approach to Mapping from Domain Requirements to Product Line Architecture." Journal of Computer Research and Development 7.

Zhu, L. and I. Gorton (2007). Uml profiles for design decisions and non-functional requirements, IEEE Computer Society.

Software Product Line - Advanced Topic

Edited by Dr Abdelrahman Elfaki

ISBN 978-953-51-0436-0

Hard cover, 122 pages

Publisher InTech

Published online 04, April, 2012

Published in print edition April, 2012

The Software Product Line (SPL) is an emerging methodology for developing software products. Currently, there are two hot issues in the SPL: modelling and the analysis of the SPL. Variability modelling techniques have been developed to assist engineers in dealing with the complications of variability management. The principal goal of modelling variability techniques is to configure a successful software product by managing variability in domain-engineering. In other words, a good method for modelling variability is a prerequisite for a successful SPL. On the other hand, analysis of the SPL aids the extraction of useful information from the SPL and provides a control and planning strategy mechanism for engineers or experts. In addition, the analysis of the SPL provides a clear view for users. Moreover, it ensures the accuracy of the SPL. This book presents new techniques for modelling and new methods for SPL analysis.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen