Modeling Variability in Product Lines Using Domain Quality Attribute Scenarios

Nadeem Abbas
Linnaeus University
Software Technology Group
Växjö, Sweden
nadeem.abbas@lnu.se

Jesper Andersson
Linnaeus University
Software Technology Group
Växjö, Sweden
jesper.andersson@lnu.se

Danny Weyns
Linnaeus University
Software Technology Group
Växjö, Sweden
danny.weyns@lnu.se

***Abstract*—The concept of variability is fundamental in software product lines and a successful implementation of a product line largely depends on how well domain requirements and their variability are specified, managed, and realized. While developing an educational software product line, we identified a lack of support to specify variability in quality concerns. To address this problem we propose an approach to model variability in quality concerns, which is an extension of quality attribute scenarios. In particular, we propose *domain quality attribute scenarios*, which extend standard quality attribute scenarios with additional information to support specification of variability and deriving product specific scenarios. We demonstrate the approach with scenarios for robustness and upgradability requirements in the educational software product line.**

***Index Terms*—variability, quality attributes, scenarios, software product lines**

I. Introduction

Variability and variability management are central in *Software Product Line Engineering (SPLE)* [1]. SPLE describes two development cycles: *domain engineering* and *product engineering* [1]. Domain engineering defines domain variability and provides a common platform for product derivation. Product engineering on the other hand exploits the variability defined by domain engineering to derive concrete products. Successful application of the product line approach to a large extent depends on how well the domain variability is defined, managed, and realized.

In this article we discuss work in progress concerning one corner stone in product line variability modeling: variability in quality attribute requirements. Modeling and managing variability in quality attribute requirements is known to be a complex problem [2]. However, clearly identified, understood, and documented requirements variability is key for the design of high quality software product line architectures. We faced the problem of variability modeling for quality concerns during the development of a software product line (SPL) for educational and research purposes (EduSPL), which is described in more detail in Section II. While eliciting and specifying EduSPL's domain requirements and their variability we found out that the available methods to model variability mainly focused on functional requirements. There is a lack of methods with explicit support to model variability of quality requirements. This is an observation we share with several others, for instance, Myllärniemi et al. [3] and Etxeberria et al. [4].

In the EduSPL project we planned to use *Quality Attribute Scenarios (QAS)* [5] for specifying quality concerns. However, standard QAS do not support modeling variability. To find a suitable method to model variability in quality attributes for a software product line we explored QAS further and studied existing techniques to model variability. This study has resulted in *domain QAS*, which extends standard QAS. We introduce fragments and parameters in the QAS template to model variability, and constraints to guide derivation of product specific scenarios. The domain QASs specify variations in quality attributes at the domain level, and are defined and documented as *core assets* during domain engineering. In product engineering such domain QASs are used to derive product specific quality attribute scenarios, i.e., *product QASs*, which support product instantiation, but can also be used for other activities, such as supporting scenario based architecture evaluations. We have applied domain QASs to specify variability for two primary quality concerns in EduSPL, robustness and upgradability. The initial results from these activities are promising, which encourages us to continue with further investigations, development, and evaluation of domain QAS.

The remainder of this article is organized as follows. The educational software product line is introduced in Section II. Section III discusses domain QASs to model product line variability in quality concerns. We exemplify domain QAS for the two scenarios of the EduSPL in Section IV. In Section V, we discuss domain QAS in a software product line engineering context. We conclude and discuss challenges for future research in this area in Section VI.

II. Educational Software Product Line (EduSPL)

The problem we address in this article was identified while developing the EduSPL. Before going into further details of the problem and the proposed solution, we first give a brief introduction of the EduSPL.

The EduSPL is a SPL for board games for educational and research purposes that is currently under development. The products of the EduSPL are game environments for multi-player board games that are deployed in a distributed setting. Game environments are expected to be available 24/7. The

EduSPL provides a complete SPL package that can be used: (1) for master projects at advanced level to study and experiment with various aspects of SPL, and (2) to support research on self-adaptation and online updates of software systems. The initial release of the SPL supports game environments with two optional two-player games. In the future, we plan to extend the asset base to support game environments with additional multi-player games.

The coarse-grained architecture of a simple EduSPL product is shown in Figure 1. This product configuration consists of two *Player Environments (PE)* and an *Operator Center (OC)* that are deployed on three hosts. Both the PEs and the OC are built from a common set of components that can be composed to the needs of the customers. Each player has access to a player environment that provides the functionality and a supporting GUI to play online games and interact with other players using a chat system. Games can be played with human players or with softbots. An operator has access to an operator center, which allows him to manage various aspects of the system, such as registration of players, performing updates, restoring player environments after failures, etc. Player environments and the operator center make use of the game platform. The game platform offers basic services for the subsystems to support system configuration, communication, security, persistence, fault handing, and dynamic updates.

EduSPL products are built on top of the OSGi platform [6], thus the various subsystems are composed of related OSGi bundles. The OSGi platform offers basic support for dynamic updates of configurations, including starting, stopping, and replacing of bundles, consistency checking of updated configurations, etc. A typical configuration of a game environment comprises 40 to 50 bundles.

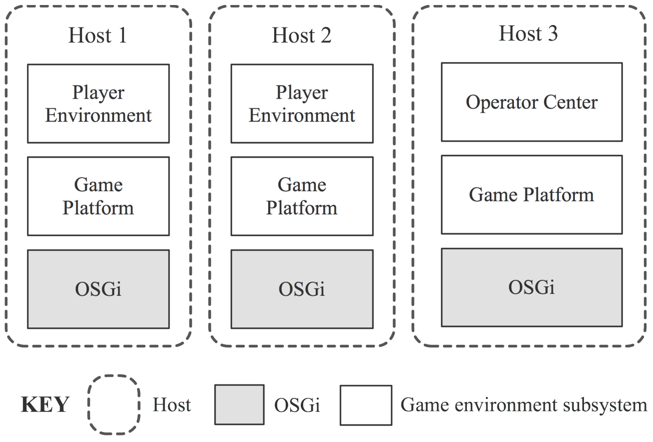


Fig. 1. Example of an EduSPL Product

Our particular interest in this paper is in quality requirements and their variability. Examples of such (informally defined) requirements are:

QR1 A new game is available and has to be added to a running game environment.

QR2 The decision algorithm of a softbot has to be replaced.

QR3 The chat system has to be upgraded.

QR4 A player environment with an ongoing game crashes unexpectedly. After rebooting the player environment, the status of the game has to be restored.

Support for these kind of requirements should be provided as services of the operator center. It is important to note that some of these requirements are only applicable for products with particular features (e.g., QR2, QR3), while others are applicable for all products (e.g., QR1, QR4). Moreover, some properties of the requirements may differ for different products. For example, the time to restore a game after a crash (QR4) may be different for products that run on different hardware platforms. Specifying requirements for each possible variant will result in huge number of requirements that are very difficult to manage. To avoid this, we need the means to express the variabilities in quality requirements.

III. Quality Attributes Scenarios and Variability

In order to support variability in quality requirements, we need the means to specify the variations and commonalities in a requirements specification. Pohl et al. [1] suggested three questions to identify requirements for product line variability:

1) *What* does vary? - e.g., an algorithm or a configuration parameter value.
2) *Why* does it vary? - e.g., support for multiple execution environments.
3) *How* does it vary? - e.g., by dynamic linking or by setting a value at compile time.

While modeling quality requirements for the EduSPL, we found out that there is no suitable approach available for modeling the quality concerns and their variations. Using standard QAS to specify quality requirements for a SPL leads to problems. In particular, for each quality concern we need to specify a large number of detailed scenarios that define concrete elements for each scenario. Basically each of the elements (source, stimulus, artifact, environment, response and response measure) may be subject of variation of a scenario. As a result, the number of scenarios domain engineers have to write to address each quality variability will be huge. Producing and managing such a large number of scenarios is cumbersome and error prone. To that end, we have investigated how QAS could be modified for this purpose.

Domain QAS

In the EduSPL project we investigated whether we could combine QAS with dedicated variability modeling techniques. This effort resulted in *domain QAS*, which is an extension of standard QAS. The rationale for extending QAS to model quality variations is the observation that the six elements in the baseline QAS can be used to answer the three 'variability questions' suggested by Pohl et al. [1]. If we group the six compartments of the standard QAS into three, as shown in Table II, they match the three questions for identifying and expressing variability of quality requirements for a SPL.

However, even if the standard QAS template allows expressing the variability, we discovered that additional information was required to completely specify the variability. We decided

Domain Quality Attribute Scenario - Template	
Compartment	**Description**
Source (SO)	This is some entity (a human, a computer system, or any other actuator) that generates the stimulus
Stimulus (ST)	The stimulus is a condition that must be considered on arrival at the system.
Artifacts (A)	Some artifacts are stimulated. This may be the whole system or some pieces of it.
Environment (E)	The stimulus occurs within certain conditions. The system may be in an 'over-load' mode or may be running in 'normal' mode when the stimulus arrives.
Response (R)	The response is the activity undertaken after the arrival of the stimulus. In the examples, we use UML scenario diagrams to model response fragments.
Response Measure (RM)	When the response occurs, it should be measurable in some fashion so that the requirement can be tested
Variants (V) & Valid QAS Configurations (VC)	*Variants* define the elements of variation of the domain QAS. *Valid QAS Configurations* define allowed combinations of the variants to derive product specific QAS from the domain QAS.
Fragment Constraints (FC)	Fragment constraints express constraints on the selection of fragments of the standard QAS compartments (i.e., Source, Stimulus, Artifact, Environment, Response, and Response Measure). There are three categories of fragment constraints: (1) *Mandatory*, (2) *Variant specific*, and (3) *Bindings*. A *mandatory* constraint defines the fragments that need to be included in all the product QAS derived from the domain QAS. A *variants specific* constraint defines the additional fragments that need to be included for a valid QAS configuration. A *binding* defines a restriction on the values of parameters for a valid QAS configuration.

TABLE I
DOMAIN QUALITY ATTRIBUTE SCENARIO

QAS Fragments	
Source	Why does it vary?
Stimulus	
Artifact	What does vary?
Environment	
Response	How does it vary?
Response Measure	

TABLE II
QAS ANSWERS THE VARIABILITY QUESTIONS

to include this additional variability modeling information in additional compartments to support understandability. The extended *domain QAS* template is described in Table I.

The first six compartments remain the same as originally defined by the standard QAS. However, we have extended the content of the compartments with *fragments* and *parameters* to support modeling the variability for a given requirement. Fragments are mandatory or optional parts of compartments that need to be selected and composed to derive concrete scenarios from the domain QAS for a product. Parameters can be used to express more fine-grained variations inside fragments. Parameters have to be bound to values for a concrete scenario.

We introduced two additional compartments (see Table I) to define the scope of the domain QAS and support deriving product specific QAS from the domain QAS, i.e., *Variants and Valid QAS Configurations* and *Fragment Constraints*. Variants define the elements of variation of the domain QAS, and valid QAS configurations define allowed combinations of the variants to derive product specific QAS from the domain QAS. Fragment constraints express constraints on the selection of fragments of the standard QAS compartments. These include mandatory fragments required for all product QAS, variant specific fragments required for specific QAS configurations, and constraints on the bindings of parameters of the required fragments.

Domain QASs are core assets that specify variations in quality concerns at the domain level. We use domain QASs to specify quality requirements for the EduSPL during domain engineering. In product engineering projects, the domain QASs are used to derive product specific QASs, i.e., product QASs, to model product specific quality requirements. Product specific scenario instantiation requires the selection of the fragments for the required valid configuration and binding parameter respecting the fragment constraints.

IV. DOMAIN QAS IN THE EDUSPL

To illustrate how the domain QAS can be used to model product line quality attributes variability, we defined two example scenarios for the EduSPL.

A. Robustness Domain QAS

The *robustness domain QAS* shown in Table III describes variability in one of the robustness requirements for the EduSPL. In particular, this requirement is concerned with restoring and resuming a failed player environment (PE) after a silent node failure (the PE becomes unresponsive after a crash). We assume that the operator center and other supporting infrastructure are operating normally during the recovery.

The *Source* and *Stimulus* compartment have one trivial fragment. The first distinct variation required is in the *Artifact* compartment, with three fragments (A1, A2, A3). *Dependent PE* represents the player environment that gets affected when the PE referred to as *A2* fails (e.g., the player of the dependent PE was playing a game with the PE that crashed).

Environment has five fragments (E1, E2,..., E5). *E1* represents an environment where a PE is online but no game or chat is ongoing. *E2* represents an environment where a PE is online with chat (but no ongoing game). *E3* represents the environment where a PE is online with an ongoing game, but no ongoing chat. *E4* represents the environment where a PE is online with an ongoing game and chat. *E5* represents the environment where a PE is playing a game against a softbot.

How the operator center responds to a detected PE failure, under different environment conditions, is characterized in the *Response* compartment using sequence diagram fragments. The first fragment (R1) specifies how the OC detects, accesses, and restores the failed PE. The three other fragments specify different variants for different conditions. *Response Measure* has only one fragment, however, it contains the *parameter* x that expresses the *range* of response time for restoring the PE supported by the EduSPL.

The *Variants and Valid Configurations* compartment characterizes three concrete failure recovery variants required in the EduSPL. The three variants V1, V2, and V3 refer to game environments with support for games, chat, and softbots. The EduSPL allows four possible QAS configurations for these variants. These four are modeled explicitly in the *Valid Configurations* part. For example, valid configuration VC3 states that the EduSPL supports game environments with only games and softbots.

Fragment Constraints define constraints on deriving product QASs for the EduSPL from this domain QAS. The mandatory constraint expresses the fragments that are required in all product QASs; e.g., R1 from the *Response* compartment. There are four *variant specific* constraints. The constraint *Variants VC1*, for instance, expresses that no additional fragments and their variants are needed for deriving the product QAS robustness is required only for ongoing games. Constraint *Variants VC3* states that the environment variant E5, and the response variant R4, must be included (in addition to the fragment specified by the mandatory constraint) in product QASs for games and softbots.

In the scenario we see how actual values are bound to the fragment parameter x. In the robustness product QASs, a concrete value for the response measure must be provided to make a scenario complete. The actual binding takes place during product engineering.

B. Upgradability Domain QAS

The *upgradeability domain QAS* shown in Table IV specifies variability in an upgradeability concern of the EduSPL. The scenario focuses on upgrades of a single PE. The variability characterized by fragments and parameters define the product QAS scope for this upgradeability concern.

How the two stimulated artifact fragments, OC and PE, responds to the two stimuli fragments under operating environment (E1) is specified by the fragments of the *Response* compartment. Fragment *R3* specifies how upgrades are downloaded and executed. The other two fragments specify how the upgrades can be initiated.

Response Measure defines three fragments of response measures. For example, *RM1* expresses that PE is notified by the OC within x seconds, where the range for x is specified as $range(x) = [5..60]$. Similarly *RM2* and *RM3* specify ranges for downloading and performing upgrades, respectively.

The *Variants and Valid QAS Configurations* compartment specifies three upgrade responses to be supported in the EduSPL. The three variants V1, V2, and V3 represent variants of QAS for *push*, *critical push*, and *pull* upgrade styles [7] respectively. A *critical push* upgrade specifies a requirement that some upgrades may not be delayed, for instance, an upgrade of a security system component.The EduSPL supports four QAS upgradeability configurations as specified by the four *Valid QAS Configurations*.

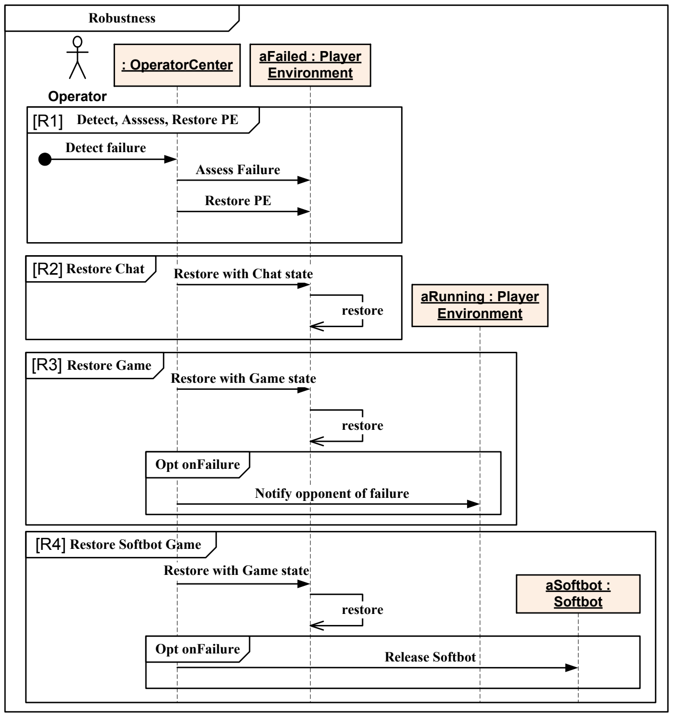
Fragment Constraints define constraints on deriving product QASs for upgradability. The mandatory constraint states that the response fragment R3 must be included in all valid configurations for product QASs. There are four *variant specific* constraints for the upgradability domain. The constraint *Variants VC2*, for example, states that fragments Stimulus-ST1, Response-R1, and Response Measure-RM1 must be included, together with the mandatory fragments, for product specific QASs that specifies support for both push (V1) and push critical (V2) upgrades.

The binding of values to fragment parameters in this scenario is concerned with specifying values for the response measure. The scenario specifies a number of constraints on time to notify, download, and upgrade. For example, bindings to VC2 and VC3 illustrate how parameter values and parameter constraints are used to specify critical upgrades. Such upgrades should not take more than 120 seconds.

V. Domain QAS in Product Line Engineering

We identified the issues related to quality concern specification addresses in this work during the *Domain Analysis* activities for the EduSPL (Section II). In the previous section we illustrated how to apply domain QAS to concrete concerns in EduSPL. The domain QAS are used in several software product line engineering activities. In Figure 2, we depict a software product line engineering process based on the process framework proposed by Pohl et al. [1]. Below we elaborate on domain quality attribute scenarios in the initial activities of domain and product engineering in the EduSPL context.

1) **Domain Analysis** – Domain analysis specifies commonality and variability in a software product line [1]. The *domain QAS* for quality concerns are considered as core assets with commonality and variability. When defining the scope for quality attribute requirements in a new

Source (SO)	**[SO1]** Player Environment (PE)
Stimulus (ST)	**[ST1]** Operator Center (OC) detects that a PE has failed
Artifacts (A)	**[A1]** OC **[A2]** Failed PE **[A3]** Dependent PE
Environment (E)	**[E1]** PE without active game or chat **[E2]** PE with active chat **[E3]** PE with active game player **[E4]** PE with active game player and chat **[E5]** PE with active game softbot
Response (R)	 Robustness Operator : OperatorCenter aFailed : Player Environment [R1] Detect, Asssess, Restore PE Detect failure Assess Failure Restore PE [R2] Restore Chat Restore with Chat state restore aRunning : Player Environment [R3] Restore Game Restore with Game state restore Opt onFailure Notify opponent of failure [R4] Restore Softbot Game Restore with Game state restore aSoftbot : Softbot Opt onFailure Release Softbot
Response Measure (RM)	**[RM1]** Restore the failed component(s) within "x" seconds, with range(x) = [120..300]
Variants (V) & Valid Configurations (VC)	**[V1]** games **[V2]** chat **[V3]** softbots **[VC1]** V1 **[VC2]** V1 $\wedge$ V2 **[VC3]** V1 $\wedge$ V3 **[VC4]** V1 $\wedge$ V2 $\wedge$ V3

Fragment Constraints (FC)	**[Mandatory]** { SO1 } $\wedge$ { ST1 } $\wedge$ { A1, A2 } $\wedge$ { E1, E3 } $\wedge$ { R1 } $\wedge$ { RM1 }
	[Bindings] RM1.bind(x)
	[Variants VC1] $\emptyset$
	[Variants VC2] { A3 } $\wedge$ { E2, E4 } $\wedge$ { R2, R3 }
	[Variants VC3] { E5 } $\wedge$ { R4 }
	[Variants VC4] { A3 } $\wedge$ { E2, E4, E5 } $\wedge$ { R2, R3, R4 }

TABLE III
DOMAIN QAS FOR ROBUSTNESS

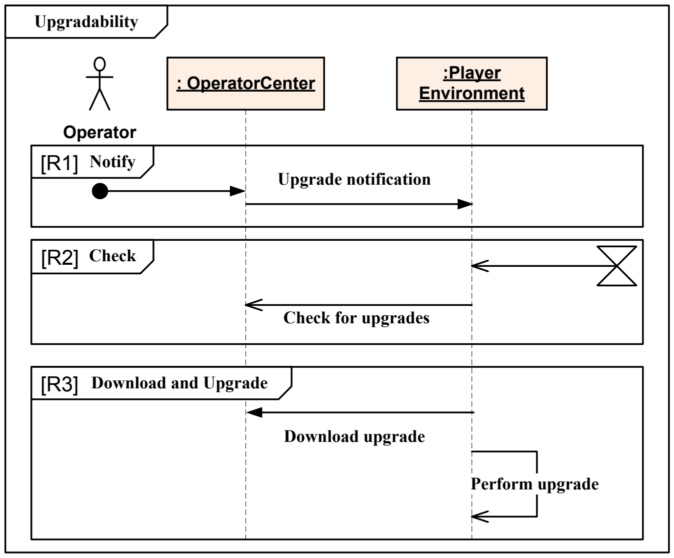
Source (SO)	**[SO1]** Operator Center (OC) **[SO2]** Player Environment (PE)
Stimulus (ST	**[ST1]** OC notifies PE about an upgrade **[ST2]** PE checks for upgrade at OC
Artifacts (A)	**[A1]** OC **[A2]** PE
Environment (E)	**[E1]** Normal operation
Response (R)	 Upgradability : OperatorCenter :Player Environment Operator [R1] Notify Upgrade notification [R2] Check Check for upgrades [R3] Download and Upgrade Download upgrade Perform upgrade
Response Measure (RM)	**[RM1]** PE notified within x seconds, with range(x) = [5..60] **[RM2]** Upgrade package downloaded within y minutes, with range(y) = [1..10] **[RM3]** Upgraded performed within z seconds, with range(z) = [15..60]
Variants (V) & Valid Configurations (VC)	**[V1]** push **[V2]** push critical **[V3]** pull **[VC1]** V1 **[VC2]** V1 $\wedge$ V2 **[VC3]** V1 $\wedge$ V2 $\wedge$ V3 **[VC4]** V3
Fragment Constraints (FC)	**[Mandatory]** { SO1, SO2 } $\wedge$ { A1, A2 } $\wedge$ { E1 } $\wedge$ { R3 } $\wedge$ { RM2, RM3 } **[Variants VC1]** { ST1 } $\wedge$ { R1 } $\wedge$ { RM1 } **[Variants VC2]** { ST1 } $\wedge$ { R1 } $\wedge$ { RM1 } **[Variants VC3]** { ST1, ST2 } $\wedge$ { R1, R2 } $\wedge$ { RM1 } **[Variants VC4]** { ST2 } $\wedge$ { R2 } **[Bindings VC1]** V1.(RM1.bind(x) + RM2.bind(y) + RM3.bind(z)) | x+y+z $\leq$ 6 minutes **[Bindings VC2]** (V1.(RM1.bind(x) + RM2.bind(y) + RM3.bind(z)) | x+y+z $\leq$ 6 minutes) $\wedge$ (V2.(RM1.bind(x) + RM2.bind(y) + RM3.bind(z)) | x+y+z $\leq$ 120 seconds) **[Bindings VC3]** (V1.(RM1.bind(x) + RM3.bind(y) + RM2.bind(z)) | x+y+z $\leq$ 6 minutes) $\wedge$ (V2.(RM1.bind(x) + RM2.bind(y) + RM3.bind(z)) | x+y+z $\leq$ 120 seconds) $\wedge$ (V3.(RM2.bind(y) + RM3.bind(z)) | y+z $\leq$ 10 minutes) **[Bindings VC4]** V3.(RM2.bind(y) + RM3.bind(z)) | y+z $\leq$ 10 minutes

TABLE IV
DOMAIN QAS FOR UPGRADABILITY

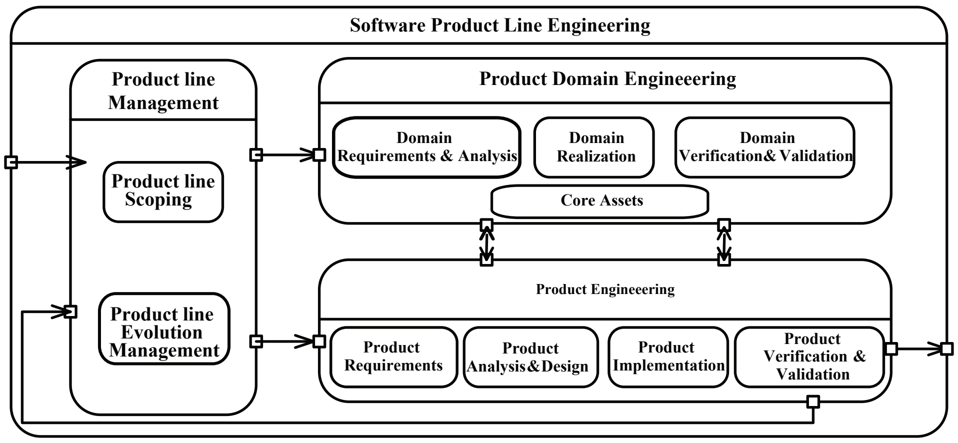

Software Product Line Engineering
Product line Management
Product line Scoping
Product line Evolution Management
Product Domain Engineeering
Domain Requirements & Analysis
Domain Realization
Domain Verification& Validation
Core Assets
Product Engineering
Product Requirements
Product Analysis&Design
Product Implementation
Product Verification & Validation

Fig. 2. Product Line Engineering (PLE) activities [1]

product-line development, for instance the EduSPL, each quality concern should be analyzed for variability and commonality. This is then specified in the different QAS compartments. The *variants* and *valid configurations* are specified together with the *QAS constraints*. During product line evolution, this activity considers *product QAS*, possibly for new quality attributes, for inclusion in the software product line as core assets. Domain QASs form the basis, together with functional and supplementary requirements for the product-line architecture, which are derived primarily during *Domain Realization*.

2) **Product Requirements & Analysis** – In a product engineering project, the *domains QAS* are core assets that provide templates to instantiate *product QAS*. Instantiation of a EduSPL product implies that specific *variants* and *valid configurations* are selected and that values are *bound* to QAS *parameters*. The product specific requirements may require modifications to the derived instances or addition of *product QAS* not covered by the product line. The rigorous expressions in the domain and product QAS may be used to verify QAS instances.
3) **Domain Realization** – The main challenge in domain realization is to derive a product-line architecture for the domain requirements, including domain quality concerns specified by domain QAS. Most architecture design methods rely on scenarios for evaluation, for example, ATAM [5]. The domain QAS plays an important role in domain design activities for the EduSPL. Initially as specifications of quality concerns that drive the architecture design method and later as the basis for concrete product QAS for scenario based evaluation. Here the rigorous underpinning of a domain QAS may be used to derive concrete *domain evaluation scenarios* based on valid configurations and fragment constraints. The application of product QAS in *product realization* is analogous to domain realization. Here the product specific QAS drive product architecture design and evaluation. The product QAS were derived from domain QAS during product analysis.

Software product line engineering will benefit from the promotion of quality attribute scenarios to core assets in *architecture documentation*, *architecture design*, and *architecture evaluation*. Above we discussed how *domain QASs* function both as domain documentation and as templates for product specific documentation. In addition, domain QAS could assist in identifying architecture patterns and tactics in architecture design and used to derive concrete scenarios for architecture evaluation.

VI. Discussion & Conclusion

In this work we address a research problem of modeling variability in quality concerns for software product lines. We identified the problem during the development of the EduSPL software product line for educational and research purposes. The contribution of this work is *domain QAS*, an approach that promotes quality attribute scenarios to core assets in software product lines. The approach reduces the number of quality attribute scenarios that must be generated as part of domain and product engineering efforts in software product line engineering. Our work on quality attributes scenarios as core assets is work in progress so evaluation is limited and several remaining challenges have been identified in the process. We discuss some below.

The primary challenge is to further extend the domain QAS to support all aspects of variability modeling for quality concerns. We have modeled examples from the EduSPL, which illustrate that the extended quality attribute scenarios approach is applicable for modeling quality concern variability in software product lines. However, these initial modeling efforts are limited. Further modeling will most certainly trigger additional work to revise the domain QAS template and its constraint language. Extensions to the template will address support for expressing variability in all QAS compartments, including improved support for fragments, parameters and their configuration and bindings. The domain QASs we have modeled so far do not exploit variability in all compartments, thus we may not say for sure that the proposed specifications approach with variants and valid configurations together with bindings and fragments is sufficient. Extensions to the constraint language, which in its current instance supports basic expressions, include several modifications to improve the constraint language's expressiveness. Using constraint languages is not novel, Karataş [8] discusses a more advance language which allows for extensive analysis. Other examples of the use of formal languages for modeling variability include the work on propositional logic by Bagheri et al. [9]. Another aspect that needs further study is the need for expressing model dependencies and other relationships, such as requires and excludes. Among the dependencies, trade-offs stand out as the most challenging. Quality attributes are non orthogonal and deriving one product QAS may impact the constraints and bindings for others. Additional challenges for the language is to incorporate modeling support to express dependencies to other model artifacts. Traceability will provide provisioning for connecting artifacts from different development activities to each other, which will assist engineers in maintaining models. The work of Tun [10] is a nice example of how to create links from requirements to products.

It is often argued that variability modeling should be factored out from the 'base' model. Variability languages such as OVM [1] and CVL [11] are two examples. [12] is an approach that extends OVM with so called meta variation points and meta variants to support addition of variants at runtime. The work of Frantz et al. [13] uses OVM with annotations. However this approach is more concerned with relating quality attributes to specific variation points and variants releated to features. Etxeberria and Sagardui [14] propose similar extensions, adding attributes to feature models to better support evaluation. For our domain QAS we decided to include variability in the specifications. The rationale was understandability and because the fragments we considered were small, which meant that they fit in one template, we argue that this approach is better from that point of view. However, it might be that additional modeling will bring fragments of such size that the approach has to be reconsidered. More complex fragments and quality concern models may require that a single concern is modeled in multiple domain QAS. However, this introduces additional dependencies that have to be supported as first-class citizen in the modeling approach.

Core assets are reused in product engineering activities. We argue that domain QASs are useful to derive product specific QAS, for instance for architecture evaluations. However, we have not yet evaluated the approach and derived product specific instances from large sets of complex domain scenarios. Efforts in that direction will most certainly raise additional issues, such as, the complexity of the derivation process, the quality of the generated scenarios, and, as a consequence, the pros and cons offered by domain QAS compared to the traditional approach of writing multiple scenarios for each quality attribute.

Finally, at this stage we have not considered any methodological support or tool support for domain QAS. However we believe that an integration with existing design and evaluation methods is important. The formal-underpinnings of domain QAS makes it a candidate for extensive tool support. We will initially focus on specification and derivation support. This may be evolved and expanded into model verification combined with advanced model management in the future.

Acknowledgments

We are grateful to Tony van Beers, Bartosz Michalik, and Quan Nguyen for their contributions to the EduSPL. We also thank the reviewers for the useful feedback that we received on an earlier version of this paper.

References

[1] K. Pohl, G. Böckle, and F. Van Der Linden, *Software product line engineering: foundations, principles, and techniques*. Springer-Verlag New York, Inc., (2005).

[2] L. Etxeberria, G. Sagardui, and L. Belategi, "Quality aware software product line engineering," *Journal of the Brazilian Computer Society*, vol. 14, no. 1, pp. 57–69, 2008.

[3] V. Myllärniemi, T. Männistö, and M. Raatikainen, "Quality attribute variability within a software product family architecture," in *Second International conference on Quality of Software Architecture QoSA*, 2006.

[4] L. Etxeberria, G. Sagardui, and L. Belategi, "Modelling variation in quality attributes," Lero The Irish Software Engineering Research Centre., Tech. Rep., 2007.

[5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Addison-Wesley Professional, (2003).

[6] OSGi Alliance. (2012) OSGi Service Platform Release 4. [Online]. Available: http://www.osgi.org/Main/HomePage. [Accessed: Jun. 21, 2012].

[7] J. Andersson, "A deployment system for pervasive computing," in *Proceedings of the International Conference on Software Maintenance*. IEEE, 2000.

[8] A. Karata, H. Ouztzn, and A. Doru, "Mapping extended feature models to constraint logic programming over finite domains," in *Software Product Lines: Going Beyond*, ser. Lecture Notes in Computer Science, J. Bosch and J. Lee, Eds. Springer Berlin / Heidelberg, 2010, vol. 6287, pp. 286–299. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15579-6_20

[9] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic, "Configuring software product line feature models based on stakeholders' soft and hard requirements," in *Proceedings of the 14th international conference on Software product lines: going beyond*, ser. SPLC'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 16–31. [Online]. Available: http://dl.acm.org/citation.cfm?id=1885639.1885642

[10] T. Than Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans, "Relating requirements and feature configurations: a systematic approach," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. Pittsburgh, PA, USA: Carnegie Mellon University, 2009, pp. 201–210. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753235.1753263

[11] F. Fleurey, O. Haugen, B. Mller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang, "A generic language and tool for variability modeling," SINTEF ICT, Tech. Rep. 978-82-14-04457-7, 2009.

[12] A. Helleboogh, D. Weyns, K. Schmid, T. Holvoet, K. Schelfthout, and W. V. Betsbrugge, "Adding variants on-the-fly: Modeling meta-variability in dynamic software product lines," in *International Workshop on Dynamic Software Product Lines*, 2009.

[13] F. Roos-Frantz, D. Benavides, A. Ruiz-Corts, A. Heuer, and K. Lauenroth, "Quality-aware analysis in product line engineering with the orthogonal variability model," *Software Quality Journal*, pp. 1–47, 10.1007/s11219-011-9156-5. [Online]. Available: http://dx.doi.org/10.1007/s11219-011-9156-5

[14] L. Etxeberria and G. Sagardui, "Variability driven quality evaluation in software product lines," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, sept. 2008, pp. 243 –252.