Integrating Quality Modeling with Feature Modeling in Software Product Lines

Joerg Bartholdt
Corporate Technology CT SE 2
Siemens AG
81739 Munich, Germany
joerg.bartholdt@siemens.com

Marcel Medak
Computer Science Dept.
Aalen University
73430 Aalen, Germany
m.medak@web.de

Roy Oberhauser
Computer Science Dept.
Aalen University
73430 Aalen, Germany
roy.oberhauser@htw-aalen.de

***Abstract*—Due to the large number of possible variants in typical Software Product Lines (SPLs), the modeling of, explicit knowledge of, and predictability of the quality tradeoffs inherent in certain feature selections are critical to the future viability of SPLs. This paper presents IQ-SPLE, an integrated tool-supported approach that considers both qualitative and quantitative quality attributes without imposing hierarchical structural constraints. The approach is demonstrated with an eHealth SPL scenario, with the results showing that this approach is promising for effectively addressing the integration of quality attributes in SPL Engineering (SPLE).**

Keywords-variability; software product lines; quality modeling; feature modeling

I. INTRODUCTION

SPLE seeks to foster a systematic reuse of software assets for different but similar software products (typically within a domain). The general approach captures the commonalities and variability of the products in the product line and splits the development into domain (commonalities) and application (additional individual features for the final product). Products are created by integrating common artifacts (usually a platform) and optionally configuring them with product-specific artifacts [6][7].

Significant work and various methodologies are well known for domain analysis and variability modeling for SPLs with a focus on features, e.g., Feature-Oriented Domain Analysis (FODA) methodology [5], FeatuRSEB [3], PuLSE [1], and others.

A significant aspect yet to be sufficiently addressed in a potentially large set of possible variants is the effect of choices on the end qualities exhibited by a variant. An SPL engineer is thus faced with many more quality-related unknowns than in a typical non-SPL architecture or application. Although various approaches for combining quality modeling with SPLE exist, previous work does not provide an integrated tool-supported approach with both qualitative and quantitative quality attributes (Q-attributes) that are explicitly considered in the variant derivation process without imposing structural constraints such as a hierarchical structure.

This paper is structured as follows: Section II describes an e-Health scenario with the ensuing requirements that initiated the research. Section III describes the IQ-SPLE solution approach, which is then evaluated in the subsequent section. Section V discusses related work, which is then followed by the conclusion and references.

II. E-HEALTH SCENARIO

For illustration purposes a simplified eHealth problem scenario that motivated this research is used. Patients are referred to other clinics due to their specialization (surgery, physical therapy, imaging, etc.). In the past, computed tomography images, clinical findings, etc. were given to the patient in the form of a printout or CD to take to the next treatment. This was error-prone, not all information was necessarily available at the next treatment location, and one was not sure that the data was the latest.

The eHealth scenario describes a clinic chain that wants to introduce SEPDE (software system for electronic patient data exchange) between organizations. The existing hospital information systems (HIS) are supplemented with SEPDE. The chain consists of ten hospitals where eight have the same HIS product and two have individual solutions.

Fig. 1 shows a reduced feature tree of the SEPDE SPL. For message processing, two message queue (MQ) variants exist with which different quality metrics can be achieved: commercial and open-source. While the commercial requires a license fee, it supports a higher throughput. Separately, the integration to an existing HIS can be done via binary protocols such as potentially more complex but faster IIOP or instead via XML-based Web Services.

For security and privacy, the secure channel (mapped to SSL connections) and the digital signature (mapped to individual message signatures) are supported. Digital signatures and a virus scan system are optional. These security measures decrease the performance, while virus scanning additionally results in ongoing costs for continuous updates.

This simplified scenario illustrates the relevance of quality attributes (e.g., performance, price, security) to the choice of specific features. The customer may not have exact requirements (e.g., 'use case 15' must be performed in less than 1.5 sec). However, the customer may be able to trade quality attributes against each other or functional features (e.g., 1.5 sec is achievable with the commercial MQ with a 15K€ license, whereas the open-source is 1.8 sec – which might be acceptable).

Each functional feature influences to some degree all system quality attributes, making the manual tracking of quality attributes difficult. Common feature trees contain

functional features that are selectable individually (with a few constraints between each other), while system quality attributes are a cross-cutting concern that changes with each (de)selection of a feature. Moreover, the quality correlations are often not expressible in simple constructs. Feature model constraints could (de)select features automatically, causing an entire set of quality attribute values to change at once.

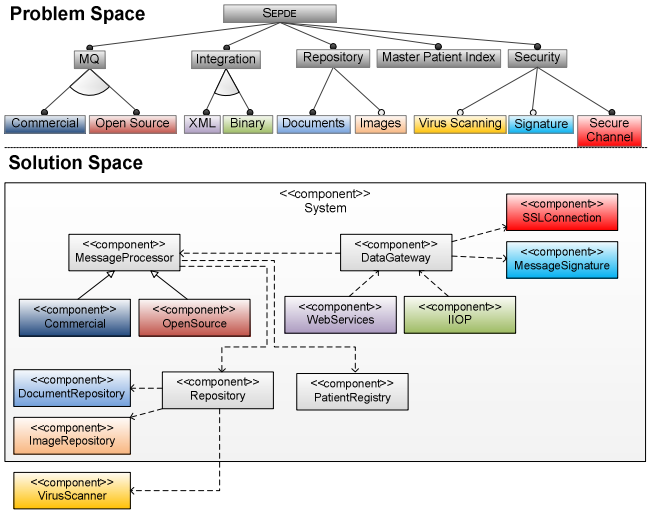

Figure 1. Reduced Feature Tree

Considering non-trivial SPLs, the impact on quality attributes are not foreseeable for the SPL engineer, thus there is the need for methodology and tool support. As quality is usually not exactly defined by the customer beforehand and requirements may change, quality support is needed during the selection process.

A. Requirements

The following requirements on the methodology (M-requirements) and tool support (S-requirements) are deduced from the scenario:

M1: *qualitative values*. It must be possible to define quality values such as low, medium, or high in cases where a quantitative expression is not feasible or would be too expensive to measure.

M2: *quantitative values*. It must be possible to define quantitative values (e.g., memory footprint, response times) in order to calculate the resulting quality values of the instantiation.

M3: *algorithms for calculating the resulting attribute values*. The methodology must support the definition of a calculation algorithm ("aggregation function") for the resulting quality value. This applies to quantitative as well as qualitative values (however, it may be less intuitive for qualitative values).

M4: *presentation as feature*. Quality values shall be presented as features in the feature tree.

M5: *customer communication*. As feature trees are also a means to communicate with the customer about the product instantiation, no implementation details shall be visible.

S1: *calculate the quality values of a given variant*. The quality attributes that result from the selection have to be calculated (ideally on-the-fly), to give immediate feedback and let the users realize what changes in quality values a change in features results.

S2: *determine the set of possible variants*. The tool shall determine the valid variants given quality constraints during the selection process.

S3: *constrain the selectable features*. Quality attribute requirements shall be definable in advance and during feature selection, with those features not selectable whose selection would impair the required quality.

S4: *visualization of quality values*. From a customer perspective, multiple quality attributes may be of interest and may differ between customers. Thus, the tooling shall support an appropriate yet configurable visualization of the resulting quality values.

III. SOLUTION

To address the aforementioned requirements, the Integrated Quality Software Product Line Engineering (IQ-SPLE) approach attaches quality attributes to the solution artifacts, maps the problem domain feature selection to the associated solution artifacts, and collects the quality attributes. With appropriate aggregation functions, the quality attributes are reduced and mirrored back to the feature tree in the problem domain for the customer to review and decide.

A. The IQ-SPLE methodology

The IQ-SPLE methodology is a 3-step process:

1) Define a list of quality attributes to be modeled and their value types (e.g., memory footprint in MB, latency in 'use case 15' in ms). This supports M1, M2, and M5. Note that discrete values need not be ordered.
2) Solution space artifacts (e.g., software components) shown in Fig. 2 are assigned quality attribute names and values. Note that each element can have multiple quality attributes assigned and these elements can be linked with features (e.g., memory consumption, use case-specific latency, scalability properties), but not all components must be assigned all attributes (components will affect memory, but not all will influence the latency in use case 15). This supports M5.

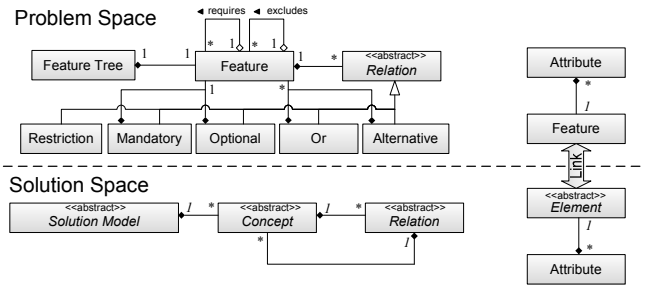

Figure 2. IQ-SPLE Meta-model

3) A quality (aggregation) function is defined to support M3 and S1. To calculate the overall quality of the resulting product instance, the quality attributes of all selected components must be aggregated. In simple cases, this can be a *sum*-operation (e.g., memory footprint, latency) or *min*-operation (e.g., security behavior is as good as the weakest component), but more complex operations are possible (e.g., encryption depends on message length which depends on the selected features, so influence may be expressed in factors such as "encryption increases latency of use case 15 by 50%"). For quality-based attributes, the aggregation function might count the number of occurrences – the most frequent wins, or calculates an average over ordered elements.

So, with this prerequisite, the quality attributes of the resulting product instance can be derived.

As mentioned above, the aggregation of quality attributes is done by mathematical functions. These functions are defined as relations between a valid variant v and a value x, where x represents the state of a specific quality.

$$q_i : v \mapsto x \tag{1}$$

To access attribute values of different artifacts, two additional functions are defined. The function *attribE* returns a single attribute value of a specific solution element (e.g., concrete component) and requires a valid variant v, a specific element e and the name of the intended attribute a.

$$attribE : v \times e \times a \mapsto x \tag{2}$$

The second function *attribV* returns a vector of all attribute values of a variant which match the provided attribute name. This provides access to values and can be used if no exceptional conditions must be taken into account.

$$attribV : v \times a \mapsto \vec{x} \tag{3}$$

Note that there are no limitations on using different value ranges for aggregations, as long as a reasonable aggregation function for the quality can be determined. Numbers for calculating memory footprint are just as possible as using low, middle, high values to assess, e.g., security aspects.

As an example, privacy could be defined as follows:

$$q_{privacy}(v) = \begin{cases} low, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 1 \\ middle, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 2 \\ high, & \sum_{\vec{x}=attribV(v,DataProtection)} x_i = 3 \end{cases} \tag{4}$$

By modeling restrictions (Fig. 2) on feature associations as constraints, M4 is supported. A constraint c_i is defined as a relation between a variant v and one element of the set *{true, false}*.

$$c_i : v \mapsto \{true, false\} \tag{5}$$

This allows a definition of, e.g., performance requirements based on predefined quality functions.

$$c_{performance}(v) = \begin{cases} true, & q_{latency}(v) \le 300ms \\ false, & else \end{cases} \tag{6}$$

The constraints are used as a way to filter out remaining variants that violate given requirements. For deselection functionality support for S2 and S3, IQ-SPLE inspects each feature beneath a selection line in the feature tree and decides if a feature selection violates the given requirements. In order to decide feature selectability, IQ-SPLE distinguishes three fundamental cases. A feature is:

a) *not selectable* if it does not occur in any remaining variant,
b) *selectable* if it occurs in every variant,
c) or *in combination selectable* if it occurs in some variants but not in all.

Cases *a* and *b* are trivial. However not every consequence of a selection can be predicted, especially if there are still open selections that affect the fulfillment of constraints. Thus IQ-SPLE uses Case *c* to indicate that a feature selection possibly can only be made dependent on further feature selections.

An example is shown in Fig. 3. To illustrate the process, a constraint is defined that forces a selection of at least five features. Initially all variants are derived that fulfill the constraint. Subfigure (1) shows the root feature tagged in green, which means that at least one variant of the feature tree matches the constraint and that feature f_0 is contained in the set of the valid variants. Green features must always be selected.

In (2) f_0 is selected and the selection cut is moved below f_0. Here f_1 is tagged in green and f_2 in red. It is obvious that f_2 will always be tagged in red because of the alternative relation to f_1, which is an invalid selection because with f_2, maximal 4 features can be selected. As a consequence, f_1 must be selected as shown in (3). In this case f_3 is selectable and f_4 in combination selectable. In (4) f_3 has been selected and f_4, f_7 and f_8 are in combination selectable. As yet the current selection doesn't fulfill the constraint of at least five selected features, but it's recognizable that a combination of the three remaining features would fulfill the constraint. If the current selection is a valid variant and fulfills the constraint, the selection of a blue feature is optional. For the case that a current selection doesn't fulfill a constraint, it is necessary to select further blue features.

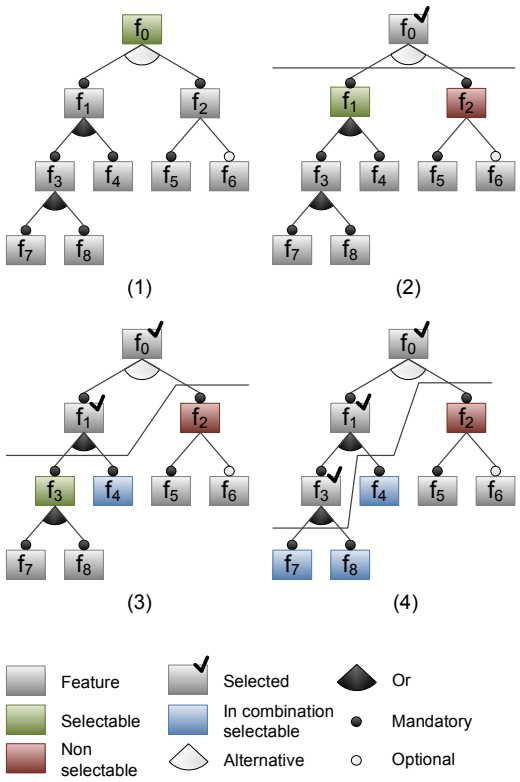

Figure 3. Selection Process Example

Thus the IQ-SPLE process supports the handling of fixed requirements and, depending on the stakeholder's perspective, see either which subset of features still fulfill the requirements or if the selection of a certain feature does not.

B. *The IQ-SPLE tool support*

An implementation of IQ-SPLE was created as an extension of the feature-oriented Eclipse modeling tool pure::variants (p::v) [8][9]. p::v provides feature and fundamental component modeling capabilities, and allows the decoration of model elements with attributes. The Component Model is used as the Solution Model of Fig. 2.

As shown in Fig. 4, the Quality Editor enables users to manage the different quality functions and presents the aggregated quality values of a variant. The Quality Evaluator calculates the quality values, which utilizes the reduced variant model created by the Model Transformer.

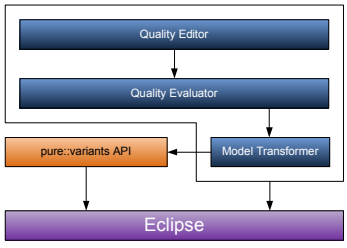

Figure 4. Quality Extension Architecture

All quality functions are defined in the Groovy language, due to the advantages of scripting and Java access. Because the direct usage of models from p::v would result in highly complicated Groovy scripts, a simplified variant model is generated from original p::v models. The complete process of computing quality values is shown in Fig. 5.

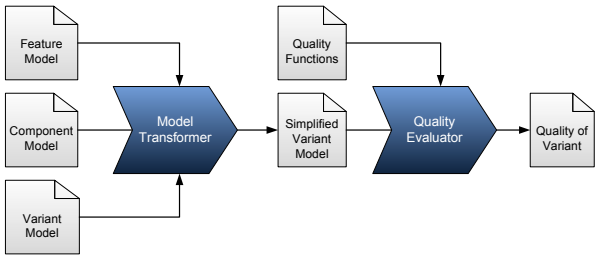

Figure 5. Quality Value Computation Process

This process is triggered every time a user changes a feature selection in p::v. The separate feature, component and variant models from p::v are transformed into a simplified variant model. After executing all quality functions by the Quality Evaluator, the resulting variant qualities with values can be presented to the user as shown in the Quality Editor of Fig. 6 (yellow ovals were appended for clarification).

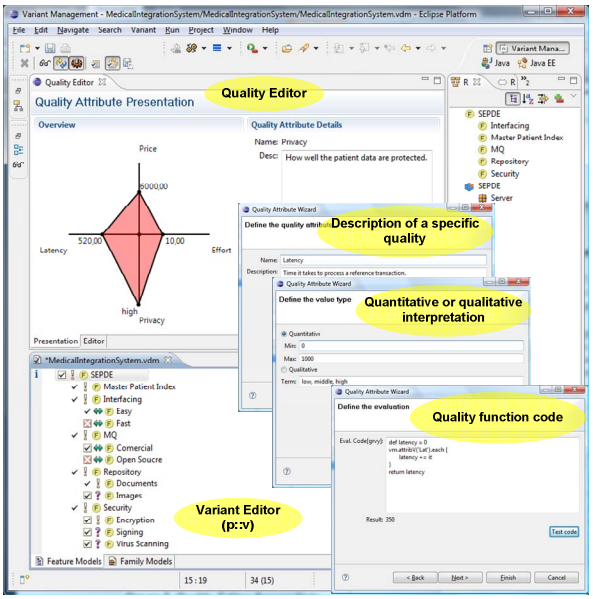

Figure 6. Quality Editor Screenshots

As mentioned in section 3.1, impacts on qualities are modeled via attributes. For the eHealth scenario, a cutout of the definition of the quality attributes is shown in Fig. 7.

As supported by p::v, the attribute values of a model element can also be dynamically calculated or become part of a variant, e.g., as a function of changes in the dependencies and selections.

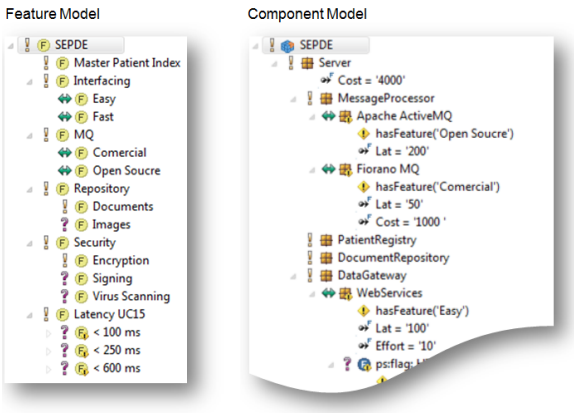

Figure 7. Models with Quality Annotations

To support S4, the assessed qualities are displayed in a spider chart (Fig. 6). The effect of a feature selection on particular qualities can be dynamically observed in the change to the chart during selection. This is helpful especially in trade-off situations.

IV. Evaluation

Evaluation criteria considered beyond the M and S requirements were an initial assessment of performance and scalability for current SPL development.

The derivation and analyses of all variants of a SPL can be computationally expensive and make its usage impractical. Thus measurements on the time behavior of the implementation for automated variant derivation were performed to determine its limits. The time to derive all variants of a feature tree with one quality constraint was measured for a binary structured feature tree of or-relations while the number of features increased from one 1 to 14. The test driver was programmed in Groovy 1.5 and was executed in Eclipse 3.4.1 with JRE 1.6 and p::v 3.0.3 on a FSC Lifebook E8210 with an Intel Core 2 2GHz and 2GB RAM running Windows Vista SP 1 with the results shown in Table I.

TABLE I. Performance vs. Features with p::v

Features	Variants	Time(s)
1	1	0.125
2	1	0.281
3	3	0.530
4	3	0.593
5	7	1.560
6	7	1.575
7	15	3.588
8	15	3.416
9	31	7.613
10	31	7.675
11	63	15.803
12	63	16.302
13	127	38.704
14	127	36.535

The number of possible variants increases exponentially with the number of features, while the performance increases roughly linearly with the number of variants. Note that real world feature trees result in significantly less variants due to exclude or XOR relations between subtrees. p::v evaluates the validity of the selection, incurring significant overhead. This overhead is unnecessary since the Quality Evaluator does this evaluation during the Quality Value Computation Process.

Due to these measurements and to see what scalability might be achievable, the implementation in Groovy was adapted to not rely on p::v APIs. The same configuration was used as above, the results of which are shown in Fig. 8. Independent of the structure of a feature tree and appearing relations, a limit in the effective usability of such an implementation without optimizations on a typical developer PC is about 100000 possible variants, taking 59.5 seconds.

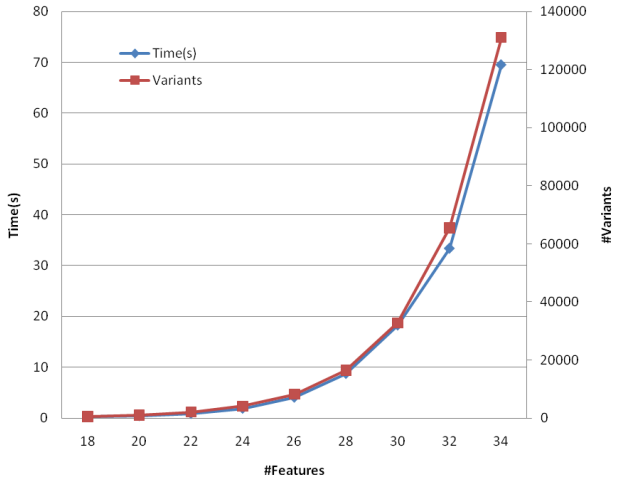

Figure 8. Native Performance vs. Features

The overhead with p::v usage of the Quality Value Computation Process for a single variant, while dependent on the Quality Function used, showed no noticeable impact to performance for linear time complexity functions. Assume there are *m* different Quality Functions, each with a linear time complexity of *O(n)*, where *n* is the number of model elements in the SPL. The overall time needed for calculation of a single variant is $m \times O(n)$, which remains linear and efficient. Inefficient evaluation of single variants can occur with the usage of inefficient Quality Functions.

V. Related Work

Related work includes the Feature-Softgoal Interdependency Graph (F-SIG) approach [4] which supports quality modeling in the domain analysis phase. Its lack of support for quantitative values results in only imprecise quality assessments of a variant. The Extended Feature Model (Ext-FM) [2] applies a Constraint-Satisfaction-Problem approach and allows both quantitative and qualitative values to determine the set of matching variants. However, it requires a hierarchical modeling of quality attributes which restricts the possible set of quality dependencies that can be modeled. The Integrated Software

Product Line Model (ISPLM) [11] integrates an implementation model that supports quantitative Q-attributes, yet it does not specify how these Q-attributes are to be utilized for a Q-assessment or set selection of variants. A comparison is shown in Table II.

TABLE II. METHODOLOGY COMPARISON

Requirement	F-SIG	Ext-FM	ISPLM	IQ-SPLE
M1: qualitative values	Yes	Yes	No	Yes
M2: quantitative values	No	Yes	Yes	Yes
M3: algorithms for calculating the resulting attribute values	No	Yes	No	Yes
M4: Presentation as feature	No	No	No	Yes
M5: Customer communication	Yes	Yes	No	Yes
S1: Calculate the quality values of a given variant	No	Yes	-	Yes
S2: Determine the set of possible variants	No	Yes	-	Yes
S3: Constrain the selectable features	No	No	-	Yes
S4: Visualization of quality values	No	No	No	Yes

COVAMOF [10] supports the modeling of dependencies between a set of variation points, however it does not explicitly address quality modeling.

VI. CONCLUSION AND FUTURE WORK

IQ-SPLE supports the eHealth scenario in several ways. First of all, the customer can make decisions about required qualities based on facts instead of subjective estimations from the SPL engineer. The customer is also able to see how a decision affects particular qualities and if the consequences of a decision are acceptable. The SPL engineer benefits since thought to system qualities are explicitly stipulated, which can help to improve the overall quality of the SPL architecture. In case a quality requirement is not fulfilled by the SPL, the engineer can track the different impacts on the quality and single out optimization opportunities. Additionally, it is possible to determine if a feature selection breaks any given quality requirements, which is done by filtering all possible variants based on existing quality requirements.

By fulfilling the M and S requirements, IQ-SPLE supports qualities in quantitative and qualitative ways. The application of constraints on features allows the explicit modeling of quality values inside a feature tree. It is not necessary to make any structural changes to feature trees or add any additional implementation details, so feature trees can still be used for customer communications. With quality functions, a mechanism is provided to transform different quality impacts into a single quality characteristic and thus make it possible to compute qualities of a variant. To make it easier to recognize how changes in feature selections affect qualities, all quality values are displayed in a spider chart.

The performance for single variant quality evaluations was sufficient for usage today, but scalability issues were found with handling large variation sets. Optimization possibilities include evaluating boundary constraints on the quality function properties to avoid further calculation overhead, e.g., aborting a calculation when a boundary value is exceeded.

Future work will also consider methods for maintaining quality attributes and synchronization of quality model attributes with other SPL-related tooling. The application of the IQ-SPLE in other industrial domains is also future work.

Models are necessarily limited in their portrayal of reality, and holistic quality modeling of a SPL due to the large set of variations faces significant challenges. While IQ-SPLE may contribute towards improved quality modeling in SPLs, much work remains.

REFERENCES

[1] J. Bayer, et al "PuLSE: a methodology to develop software product lines", In Proceedings of the 1999 Symposium on Software Reusability (SSR '99), ACM, pp. 122-131, 1999.

[2] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models", in LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, pp. 491–503, 2005.

[3] M. L. Griss, J. Favaro, and M. d. Alessandro, "Integrating feature modeling with the RSEB," Proc. of the 5th International Conference on Software Reuse (1998), ICSR, IEEE Computer Society.

[4] S. Jarzabek, B. Yang, and S. Yoeun, "Addressing quality attributes in domain analysis for product lines," IEE Proceedings Software, vol. 153, no. 2, pp. 61–73, 2006.

[5] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Carnegie Mellon University, 1990.

[6] F.J. v.d. Linden, K. Schmid, and E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, 2007, ISBN 3540714367.

[7] K. Pohl, G. Böckle, and F.J. v.d. Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Springer, 2005, ISBN 3540243720.

[8] pure-systems GmbH, Variant Management with pure::variants: Technical White Paper, 2006.

[9] pure-systems GmbH, pure::variants www.pure-systems.com/Variant_Management.49.0.html [June 13, 2009]

[10] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Modeling dependencies in product families with COVAMOF", Proc. of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006), March 2006.

[11] N. Siegmund, M. Kuhlemann, M. Rosenmüller, C. Kästner, and G. Saake, "Integrated product line model for semi-automated product derivation. Using non-functional properties," Proc. of the Workshop on Variability Modelling of Software-intensive Systems (VaMoS), pp. 25–31, 2008.