Quality Controlled Composition Generation

Jan Reimann

Technische Universität Dresden
Institut für Software- und Multimediatechnik
D-01062, Dresden, Germany
jan.reimann@tu-dresden.de

Abstract. Software development processes include functional and non-functional requirements (NFRs) in the analysis phase but usually do not propagate NFRs into the design and implementation. In many cases NFRs are tackled after implementation for performance tuning. However, software as needed in cyber-physical systems is highly dependent on *quality requirements* (which we use synonymously for NFRs), because they interact with the physical world and safety aspects are critical. Thus, quality concerns must drive development processes in the whole life cycle. Component-based software engineering gives a good starting point for integrating NFRs into the whole development process, since implementations satisfy well-defined interfaces and enable variability referring to the ability of exchanging a component's implementation. This variability suggests that different implementations can be distinguished in terms of the quality requirements they fulfil in contrast to their functional properties. Hence, NFRs are a variability dimension in software systems which must be considered. In this paper we outline an approach combining hyperspaces, enabling separation of quality concerns, with feature trees, being used for quality variability handling, for enabling explicit quality modelling to drive system composition. Dependent on different system contexts then a new composition can be generated based on the quality requirements which are needed to be satisfied in certain contexts.

Keywords: Composition Generation, Quality Variability, Hyperspace, Feature Modelling, Cyber-Physical System, Quality Controlled Composition

1 Motivation

Cyber-physical systems (CPS) are a good example for quality-dependent systems because software and physical processes affect each other in feedback loops [9]. Hence, unforeseeable changes of the physical world are propagated to the virtual world and therefore cannot be predicted completely at design time. For this reason, quality requirements such as response-time, safety and energy consumption must underlie special attention to prevent self-violation of the system and its surrounding. Thus, context changes influence the quality requirements of the system and must trigger a regeneration of the overall system composition

because different contexts may need varying quality requirements. In this sense, the whole system can be composed optimally w.r.t. the current context.

In quality aware systems three main characteristics need to be satisfied: 1) *multi-qualities*, 2) *quality dependencies*, and 3) *dynamic composition generation*. In the following we discuss the problems associated with those characteristics.

Since quality requirements cannot be fulfilled by a single component [13], quality concerns are widespread over the whole architecture without being connected to each other conceptually. This non-related distribution of quality handling results in a system which is not dynamically recomposed if contexts change and therefore other quality requests must be served. This constitutes the problem that no system knowledge exists about which artefacts fulfil which quality requirements because of unknown scattered quality requirement realisations.

Qualities are interrelated and may not be considered in isolation [5]. That means, if the context changes along with the context's quality requirements it must be clear which dependent quality requirements must be satisfied although they are not directly affected by the context change.

Dynamic generation of system compositions dependent on context changes require for being able to navigate from quality requirements to satisfying artefacts at design time and at runtime. Only runtime tracing from quality requirements to fulfilling components exploit the knowledge of which quality requirements hold in a certain context and is therefore essential in the whole development process.

2 Hyperspace Quality Controlled Composition System

We believe that the general strategy to tackle the described problems is a separation of quality concerns instead of scattering them over the system. To separate concerns we must specify quality attributes and functional artefacts detached from each other. The basic idea is to build on the Hyperspace approach with multi-dimensional separation of concerns from Ossher and Tarr [11] because qualities are crosscutting and can be handled if considered as dimensions in a hyperspace.

As a precondition we distinguish between *quality requirements* of the system under development in contrast to *quality assertions*, being satisfied by a certain artefact. Quality requirements are detached from artefacts because they don't make assumptions about the concrete components contained in the system. On the other hand quality assertions express explicitly which concrete quality values can be fulfilled by units and therefore are connected to them. Fig. 1 a) illustrates an example with the quality assertions *response time*, *energy consumption* and *accuracy* for exemplary concrete component implementations encapsulated behind the two interfaces *car control component* and *distance sensing component*.

Assuming that all components possess quality assertions, we span up a hyperspace. As can be seen in Fig. 1 b) every quality complies with one dimension. Each dimension is divided into concerns w.r.t the quality assertions from the

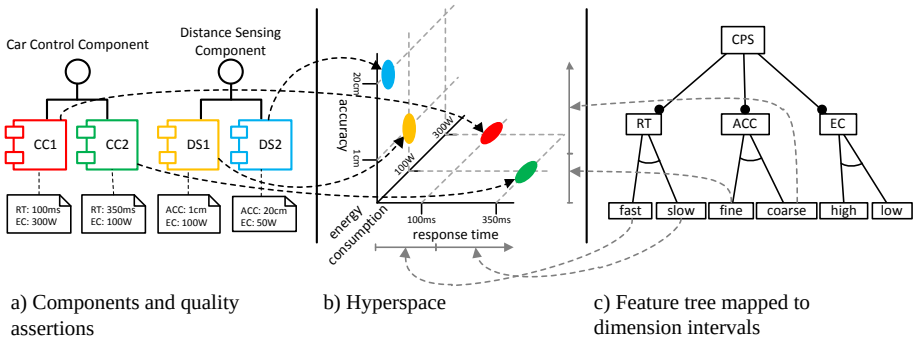

Fig. 1. Indexing the hyperspace with quality requirements by mapping them to dimension intervals (RT = response time, EC = energy consumption, ACC = accuracy)

components. Thus, each artefact is arranged into the hyperspace by considering the quality assertions as coordinates on each dimension. Furthermore, quality assertions may not only be concrete values but functions. This way, artefacts are arranged into the hyperspace by means of point sets. Thus, the mapping can be non-injective. However, without an index mechanism for granting access, we cannot benefit from the hyperspace because the artefacts need to be located.

The idea is that all quality requirements of the system are considered as features in a feature tree. Each quality requirement complies with a feature and its classifications are subfeatures. To achieve an effective index mechanism of the hyperspace we now map the quality features from the tree to intervals of the hyperspace. This mapping results in a semantic connection between the decoupled quality assertions of the artefacts and the quality requirements of the system. If, e.g., energy consumption, which was stated in the requirements engineering phase, is divided into *high* and *low* we now can determine which components in the hyperspace comply with those quality requirements. This relation is depicted in Fig. 1 b) and 1 c). In the right part the quality requirements can be seen mapped to the corresponding dimensions. To further express dependencies between quality requirements we consider approaches such as described in [2] where propositional formulas specify fine-granular constraints between different features which cannot directly be read from the feature model. With such propositional formulas it is possible to specify, e.g., that fine accuracy implies a fast response time. Furthermore, fast response time can imply high energy consumption. If a context change now results in the change of the quality requirement of accuracy from coarse to fine, it can automatically be derived that not only the implementation of the distance sensing component must be exchanged but also the implementation of the car control component. These quality dependencies allow for a flexible mechanism of determining which quality requirements are affected by context changes and therefore which artefacts have to be exchanged or modified. Thus, a new system composition can be generated automatically for changing contexts.

3 Related Work

The only known combination of the Hyperspace approach with feature trees is the HyperFeatuRSEB method [4]. Regarding to [14] features compose one dimension and requirements specified in use-cases compose the other to derive a system's architecture. Limitations of HyperFeatuRSEB are that qualities are not considered and that the methodology is too static in terms of only having two fixed dimensions.

Research in the field of specifying qualities explicitly has been done in [1] which resulted in the *Component Quality Modeling Language (CQML)*. Later on, an extension emerged named *CQML+* [12] and finally the *Energy Contract Language (ECL)* was developed as an extension of the formerly mentioned w.r.t. energy consumption [7]. All of them specify quality contracts between software components. In addition, Götz et al. defined a metamodel for ECL which we plan to reuse and improve for our approach since we want to promote model-driven development of quality aware systems.

Finally, approaches exist in [3,6,10,13] which explicitly consider qualities as features. However, all of them don't separate quality requirements from quality assertions.

4 Research Method

To proof the proposed approach the concepts must be evaluated by several examples. In a first step it is essential to design a CPS scenario. Therein we plan to use the humanoid robot NAO, manufactured by Aldebaran[1], which has sensors and actuators and thus forms a CPS, since it can interact with the physical world and communicate with the cyber world such as cloud services. The second step is to use our concepts to specify and implement this scenario to approve our approach. More CPS scenarios will follow then. Furthermore, research has to be done in the areas on how quality assertions can be measured and guaranteed, how contexts can be modelled and related to quality requirements, how this approach can benefit from context-oriented programming [8], and how contexts can parameterise quality dependencies. For the latter, a good starting point would be to analyse first-order logic to be feasible since predicates can be defined which then are usable to specify interrelations between qualities as a function of a context. Crosscutting to the above research directions, a good tooling platform must be implemented to support development of quality aware systems for which a new system composition can be generated at runtime.

Acknowledgements

This research is inspired and supervised by Prof. Dr. rer. nat. habil. Uwe Aßmann and is co-funded by the European Social Fund and Federal State of Saxony within the project ZESSY #080951806.

[1] `http://www.aldebaran-robotics.com/`

References

1. Aagedal, J.O.: Quality of Service Support in Development of Distributed Systems. Ph.D. thesis, University of Oslo, Norway (2001)
2. Batory, D.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) Software Product Lines, Lecture Notes in Computer Science, vol. 3714, pp. 7–20. Springer Berlin / Heidelberg (2005)
3. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Pastor, O., Falcão e Cunha, J.a. (eds.) Advanced Information Systems Engineering, Lecture Notes in Computer Science, vol. 3520, pp. 381–390. Springer Berlin / Heidelberg (2005)
4. Böllert, K.: Objektorientierte Entwicklung von Software-Produktlinien zur Serienfertigung von Software-Systemen. Ph.D. thesis, Technische Universität Ilmenau (2002)
5. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers Boston/Dordrecht/London (2000)
6. Etxeberria, L., Sagardui, G.: Variability driven quality evaluation in software product lines. In: Proceedings of the 2008 12th International Software Product Line Conference. pp. 243–252. IEEE Computer Society, Washington, DC, USA (2008)
7. Götz, S., Wilke, C., Schmidt, M., Cech, S., Aßmann, U.: Towards energy auto tuning. In: Proceedings of First Annual International Conference on Green Information Technology (GREEN IT) (2010)
8. Hirschfeld, R., Costanza, P., Nierstrasz, O.: Context-oriented programming. Journal of Object Technology 7(3), 125–151 (March-April 2008)
9. Lee, E.A.: Cyber physical systems: Design challenges. Tech. Rep. UCB/EECS-2008-8, University of California at Berkeley, Electrical Engineering and Computer Sciences (January 2008)
10. Myllärniemi, V., Männistö, T., Raatikainen, M.: Quality attribute variability within a software product family architecture. In: Second International Conference on the Quality of Software Architectures (QoSA) (2006)
11. Ossher, H., Tarr, P.: Multi-dimensional separation of concerns and the hyperspace approach. In: Kluwer (ed.) Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development (2000)
12. Röttger, S., Zschaler, S.: Cqml+: Enhancements to cqml. In: Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering. pp. 43–56. Cépaduès-Éditions, Toulouse, France (2003)
13. Sincero, J., Schröder-Preikschat, W., Spinczyk, O.: Approaching non-functional properties of software product lines: Learning from products. In: APSEC. pp. 147–155 (2010)
14. Sochos, P., Philippow, I., Riebisch, M.: Feature-Oriented Development of Software Product Lines: Mapping Feature Models to the Architecture. In: Object-Oriented and Internet-Based Technologies, pp. 23–42 (2004)