Modeling of Quality Attributes using an Aspect-Oriented Software-Product Line Approach

José Miguel Horcas

CAOSD Group, University of Málaga, Spain
horcas@lcc.uma.es,
WWW home page: http://caosd.lcc.uma.es/

Abstract. Modeling Functional Quality Attributes (FQAs) and the relationships between them is a complex task for several reasons. In order to cope with that complexity, our research focuses on defining an aspect-oriented software product line approach that: (1) models the commonalities and variabilities of FQAs from the early stages of the development process using a software product line approach, (2) weaves the FQA configurations with the core functionality of applications in an aspect-oriented fashion, (3) specifies the FQAs in a generic way, making it possible to use our approach with any MOF-compliant modeling language, and (4) adapts the FQA configurations at runtime according to changes in the application execution environment.

Keywords: functional quality attributes, variability, SPL, AOSD, MDD, dynamic reconfiguration

1 Introduction

Quality attributes (QAs) are overall factors that affect the design, run-time behavior and user experience of a software system (e.g., usability, performance,...) [1]. These factors must be well understood and articulated early on in the development process. The list of documented QAs is very large and some of them have strong functional implications that can be easily modeled by software components (e.g., error handling, security,...). Others, such as cost, efficiency, portability, etc., could be mapped to architectural or implementation decisions, but not directly to functional components. We define Functional Quality Attributes (FQAs) as those QAs that have clear implications for the functionality of a system. So, for these FQAs there are specific components (e.g., a component implementing an encryption algorithm) that need to be incorporated into the software architecture of the system in order to satisfy application requirements. Examples of FQAs are security, usability, persistence and context awareness.

Modeling FQAs is not a straightforward task for several reasons. On the one hand, they are usually very complex, being composed by many concerns and making the task of modeling them from scratch hard and error prone. For example, security is a FQA composed by authentication, access control, encryption and non-repudiation concerns, among others. This complexity is further

complicated by the fact that each software system requires a variable number of concerns. For instance, one application may require the authentication and encryption security concerns while another application may require the non-repudiation and the privacy security concerns. Moreover, the concerns that comprise a given FQA have dependencies and interactions with each other, and also with other FQAs' concerns. For example, the confidentiality concern will depend on the authentication, encryption and access control concerns, while the security concerns are required to satisfy other FQAs like, for example, usability, adaptability or context awareness. On the other hand, a FQA may be required in various points of the same application (e.g., security needs to be usually ensured in different points of an application) and thus their functionality is usually tangled and/or scattered with the core functionality of the application. Finally, FQAs may need to be adapted at run-time due to changes in the execution environment of the applications (e.g., user preferences, quality negotiation, limited battery/memory/CPU in mobile applications).

In spite of their importance, these issues are not all appropriately taken into account by existing approaches [2, 3]. For instance, both proposals are very dependent on the architecture description language and the variability language used throughout the approach, which complicates the extension of the approach to other domain specific languages. [3] does not even consider the existing dependencies between the different FQAs. Another limitation of these proposals is that they only deal with static configuration of the FQAs without addressing the problems of the dynamic adaptation of the FQAs at runtime. Thus, in order to address all the aforementioned challenges, our research focuses on modeling FQAs by combining the benefits provided by several technologies, such as Software Product Lines (SPLs) [4], Aspect-Oriented Software Development (AOSD) [5] and Model-Driven Development (MDD) [6]. On the one hand, the use of a SPL approach allows us to manage the complexity and the variability of the FQAs from early stages of the development. Moreover, making the SPL models available at runtime ('models@runtime' in Dynamic SPLs [7]) our approach provides support for reconfiguring FQAs at runtime. On the other hand, the use of AOSD allows us to achieve a better separation of the FQAs crosscutting concerns. Finally, we use MDD in order to make our proposal as generic as possible. Concretely, the specification of FQAs will be completely independent from the language used to model the base applications, being the only requirement the use of a MOF-compliant modeling language. Therefore, the generic models of the FQAs defined in a language such as UML [8] will be automatically transformed to concrete models defined in the language used to model the base application.

Following this introduction, Section 2 presents the problem description, in more detail. Then, in Section 3 we discuss the main contributions of our proposal. Section 4 describes the related work, comparing it with our approach, and finally Section 5 presents the conclusions.

2 Problem Description

As previously stated, modeling FQAs (e.g., security, usability, persistence, etc.) is not a straightforward task for several reasons. In this section we discuss the main challenges that need to be taken into account when modeling FQAs.

Challenge 1. Manage the complexity of FQAs. Most FQAs are very complex, as they are composed by many concerns. The security FQA, for example, is composed by confidentiality, integrity, access control, authentication, privacy, encryption, non-repudiation, recovery, among many other concerns. Moreover, these concerns have dependencies and interactions with each other, such as the confidentiality concern that depends on the authentication, encryption and access control concerns. Furthermore, FQAs also have dependency relationships with other FQAs. For instance, the contextual help concern of the usability FQA depends on the authentication concern of the security FQA in order to provide customized help based on the previous experience of the user with a particular application. These dependency relationships between concerns of different FQAs often go unnoticed by the software architects, who are not domain experts in modeling FQAs.

Challenge 2. Model adaptable and reusable FQAs solutions. FQAs are recurrent and have many points of variability. They are recurrent in the sense that the same FQAs are required by several applications. However, there are many variation points because not all of the concerns of a FQA are required by all the systems. Thus, functionality that is not required by the final application, and that will be never used, should not be incorporated into the final application. This means that the models of the FQA need to be customized to the requirements of each application. In addition, different configurations of the same FQA may be required in different parts of an application. So, the variability models also need to consider multiple configurations of the same FQA.

Challenge 3. Define a generic and language-independent solution. The applications that require the FQAs may be specified using different modeling languages. Consequently, the definition of the FQAs must be as generic and language-independent as possible. Otherwise, the benefits of modeling these FQAs in an adaptable and reusable way would be reduced or even lost. However, after generating a customized configuration of the FQAs, the resulting models need to be combined with the models of the application, which are specified using a particular modeling language. Thus, in order to do that, a set of model transformations are needed in order to obtain specific models of the FQAs (i.e. models specified using a particular modeling language) from the generic models of the FQAs.

Challenge 4. Achieve separation of concerns between FQAs and the core applications. Most FQAs are crosscutting concerns that need to be present in several parts of a system. This means that the concerns are normally scattered (i.e. the same concern is present in more than one software module) and/or tangled (i.e. the same module includes more than one concern) with the base functionality of an application. For example, the encryption concern is required by the components that both send and receive encrypted data, and

thus the encryption behavior is scattered among these components, and tangled with their base functionality. Another example is the feedback concern of the usability FQA, which crosscuts all the points of the application in which some kind of feedback information needs to be provided to the user.

Challenge 5. Provide support for the runtime adaptation of FQAs. Applications that run in highly dynamic environments continuously change their requirements at runtime. If the new requirements affect the FQAs, they need to be dynamically adapted. Examples of these applications can be mobile applications that need to be adapted to changes in their environment. For instance, a user moves from a secure to an unsecured environment and an encryption concern need to be incorporated to his/her mobile applications in order to encrypt the communications and make them more secure. Another example can be mobile or desktop applications that need to negotiate their qualities in order to correctly interoperate with other parties. For instance, using a trust-negotiation protocol in which the security policy used by two parties in a distributed communication is negotiated, and the security concerns used by the application will change depending on the results of the negotiation.

3 Our proposal

In order to achieve solutions for all these challenges, we propose an Aspect-Oriented Software Product Line (AO SPL) approach to model the FQAs from the early stages of the software development process in a generic and language-independent way. Figure 1 shows an overview of our approach that combines the SPLs, AOSD and MDD software technologies.

Firstly, in order to manage the complexity and the variability of the FQAs from the early stages of the development we use an SPL approach. An SPL allows us to create and maintain a collection of FQAs and to customize them for each application. In our approach, we specify the FQAs generically (top of Figure 1). We use a generic variability model (e.g., a model specified in the Common Variability Language (CVL) [9]) to express the commonalities and variabilities of the FQAs and any MOF-compliant modeling language (e.g., AO-ADL [10]) to define the functional behavior of the QAs in architectural models. FQAs are modeled only once, creating architectural patterns with reusable FQAs solutions (e.g., architectural templates [11]).

Secondly, due to the fact that different applications are specified using different specific modeling languages, the generic models of the FQAs previously specified need to be adapted in order to customize them for each application. To do this, we propose defining a set of model transformations to obtain concrete models specified in the particular modeling language used by the application[1] from the generic models of the FQAs (centre of Figure 1). Then, the concrete models of the FQAs can be customized to the requirements of that particular application. A customized configuration of the FQAs is generated in the specific modeling language of the base application.

[1] The modeling language of the application will be based on MOF metamodels.

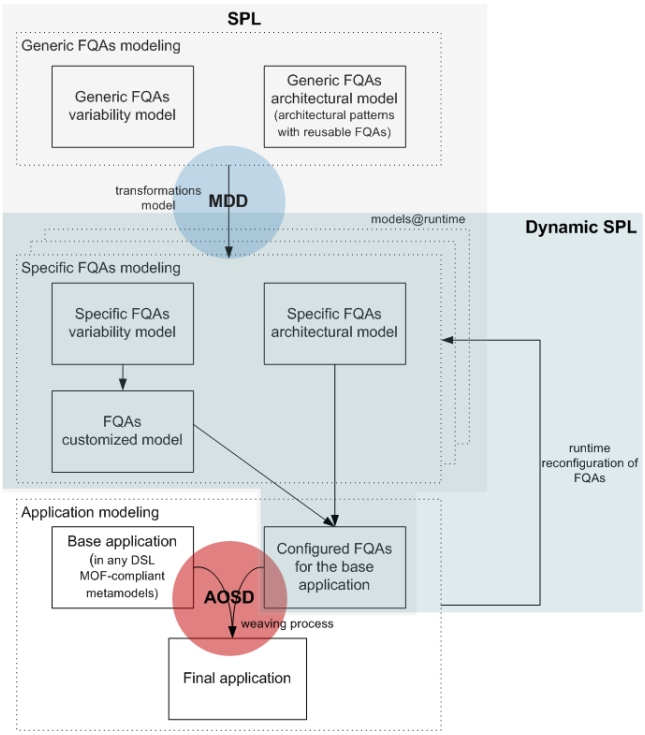

Fig. 1. Our AO SPL approach.

Then, once a configured architectural model of the FQAs has been obtained we need to incorporate it into the base application. We propose to do this, using AOSD (bottom of Figure 1). AOSD enables us to weave the architectural model of the FQAs with the core software architecture of the base application non-intrusively — i.e., without modifying the existing components in the software architecture of the base application.

Finally, as explained in the previous section, the requirements of the application may change at runtime. So, we use a Dynamic SPL by making the specific models of the FQAs available at runtime in order to allow the dynamic reconfiguration of the FQAs in response to changes in the application's execution environment.

4 Related Work

There are quite a number of approaches that model variability, some of them expressing the variability by feature models such as in [12], and others expressing the variability in relation to a base model (e.g., with annotations and/or using a variability language), such as in [13, 14]. However, there has been little work

devoted to model the variability of QAs which are strong functional implications and need to be modeled as functional software components.

We tried to address some of the problems described in Section 2 in previous approaches [2, 3] by using AO-ADL [10] as modeling language and VML [15] as the variability language. However, the solution that we proposed is totally dependent on both the modeling language and the variability language. This is because the VML directly depends on the modeling language used to describe the FQAs and the architecture of the application (AO-ADL in this case). So FQAs cannot be reused in applications defined in other modeling languages. Moreover, FQAs are configurable only before deployment in the software systems and therefore dynamic reconfiguration cannot be achieved.

There are also some approaches that use a variability language such as VML. [15] defines another way of obtaining a software architecture from a feature model. VML is used to describe different variation points with actions to be applied on an existing architecture following a positive approach that builds architectures by adding new components and connectors to a core architecture. [16] is a different approach that automatically generates a VML for different feature models and architectural languages. This generation has a fairly automated process by using a plug-in for that language, although the process requires the inclusion of the feature model and the final ADL metamodels.

An interesting approach to model quality attributes and their dependency relationships is presented in [17] and [18]. Rasha Tawhid and Dorina Petriu propose a technique to model the commonality and variability in structural and behavioral SPL views using MDD. The proposal adds generic annotations related to a QA (e.g., performance) to a UML model that represents the set of core reusable SPL assets. Then, through model transformations, the UML model of a specific product with concrete annotations of the QA is derived, and a model for the given product is generated. However, this approach models variability expressed in relation to a base model, they extend the used base modeling language (e.g., UML profiles with stereotypes) to annotate the base model with elements that are subject to variability. Annotating the base model makes this extremely related with variability specifications and avoids the reuse of both the base model of the application and the model of the QAs. Other important differences with our proposal are: (1) they model non-functional QAs such as performance instead of FQAs; (2) they introduce the variability at the design level (e.g. within sequences diagrams) while we model the variability of the QAs earlier on in the development process, at the architectural level; (3) their approach uses MDD, focusing on the analysis of the non-functional QAs in the final product, and we use MDD to generalize and make the specification of the FQAs completely independent from the language used to model the base application.

As variability modeling is often closely associated with product lines, most of the approaches use an SPL in their proposals such as [19], [20], and [21]. Although none of these papers focus on modeling the FQAs and the existing relationships between them, they do address variability at the architectural level

using SPL. In [19], the authors propose a mechanism based on the principles of Invasive Software Composition techniques in order to explicitly specify the commonalities and variabilities of SPLs at the architectural level without tangling the core and product architectures of the SPL. In [20], McVeigh et al. propose an approach based on component resemblance in which base components can be modified in order to reuse them without affecting the rest of the users of the component. The changes are not included in the base components but in new components created from them, resulting in an inheritance-like approach. This proposal addresses the reuse problem successfully but it is not efficient in the sense that all the common functionality is also replicated, which can introduce a considerable overhead. Finally, a hierarchical variability modeling mechanism is proposed in [21]. Variation points are defined inside the components, and the different variants specify how they are configured. This approach is supported by a variability metamodel and a tool based on the MontiArc ADL.

Finally, much related work has been done to explore possible solutions to address runtime variability, as in [22–26]. But, none of it includes reconfigured quality attributes at runtime.

5 Conclusions

Our research focuses on defining an aspect-oriented software product line approach that combines SPLs, AOSD and MDD software technologies in order to model FQAs. We use SPLs to model the commonalities and variabilities of FQAs from the early stages of the development process. We make our proposal as generic as possible using MDD in order to achieve a better reuse of the FQAs. We specify the FQAs in a generic way and customize them for applications that are defined in any MOF-compliant modeling language. The FQA configurations are weaved with the core functionality of applications in an aspect-oriented way. AOSD allows us to achieve a better separation of the FQAs crosscutting concerns. Our approach also supports the dynamic reconfiguration of the FQAs by using a Dynamic SPL and models@runtime.

Acknowledgment

This research has been conducted in collaboration with Mónica Pinto and Lidia Fuentes from the Languages and Computer Science Department at the University of Málaga, which are the PhD Supervisors of the thesis discussed in this paper. Work supported by the European Project INTER-TRUST 317731 and the Spanish Projects TIN2012-34840 and FamiWare P09-TIC-5231.

References

1. Barbacci, M., Klein, M., Longstaff, T., Weinstock, C.: Quality Attributes. Technical Report CMU/SEI-95-TR-021 ESC-TR-95-021, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania (1995)

2. Horcas, J.M., Pinto, M., Fuentes, L.: Variability and dependency modeling of quality attributes. In: Software Engineering and Advanced Applications (under revision). (2013)
3. Lence, R., Fuentes, L., Pinto, M.: Quality attributes and variability in AO-ADL software architectures. In: Proceedings of the 5th European Conference on Software Architecture: Companion Volume. ECSA '11, New York, NY, USA, ACM (2011) 7:1–7:10
4. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: foundations, principles and techniques. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2005)
5. aosd.net: Aspect-Oriented Software Development. http://www.aosd.net/
6. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer **39**(2) (February 2006)
7. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines. Computer **41**(4) (April 2008) 93–95
8. OMG: Unified Modeling Language (UML), Infrastructure, V2.1.2. Technical report, OMG (November 2007)
9. Haugen, O., Wąsowski, A., Czarnecki, K.: CVL: Common Variability Language. In: Proceedings of the 16th International Software Product Line Conference - Volume 2. SPLC '12, New York, NY, USA, ACM (2012) 266–267
10. Pinto, M., Fuentes, L., Troya, J.M.: Specifying aspect-oriented architectures in AO-ADL. Information and Software Technology **53**(11) (2011) 1165–1182
11. Pinto, M., Fuentes, L.: Modeling quality attributes with aspect-oriented architectural templates. J. UCS **17**(5) (March 2011) 639–669
12. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (1990)
13. Fontoura, M., Pree, W., Rumpe, B.: The UML profile for framework architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)
14. Gomaa, H.: Designing Software Product Lines with UML 2.0: From use cases to pattern-based software architectures. In: Software Product Line Conference, 2006 10th International. (2006) 218–218
15. Loughran, N., Sánchez, P., Garcia, A., Fuentes, L.: Language support for managing variability in architectural models. In: Proceedings of the 7th international conference on Software composition. SC'08, Berlin, Heidelberg, Springer-Verlag (2008) 36–51
16. Zschaler, S.: VML*: A generative infrastructure for variability management languages (2009)
17. Tawhid, R., Petriu, D.: Integrating performance analysis in the Model Driven Development of Software Product Lines. In Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M., eds.: Model Driven Engineering Languages and Systems. Volume 5301 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2008) 490–504
18. Tawhid, R., Petriu, D.: Automatic derivation of a product performance model from a software product line model. In: Software Product Line Conference (SPLC), 2011 15th International. (2011) 80–89
19. Perez, J., Diaz, J., Costa-Soria, C., Garbajosa, J.: Plastic partial components: A solution to support variability in architectural components. In: Software Architecture, 2009 European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on. (September 2009) 221–230

20. McVeigh, A., Kramer, J., Magee, J.: Using resemblance to support component reuse and evolution. In: Proceedings of the 2006 conference on Specification and verification of component-based systems. SAVCBS '06, New York, NY, USA, ACM (2006) 49–56
21. Haber, A., Rendel, H., Rumpe, B., Schaefer, I., van der Linden, F.: Hierarchical variability modeling for software architectures. In: Software Product Line Conference (SPLC), 2011 15th International. (August 2011) 150–159
22. Zdun, U.: Dynamically generating web application fragments from page templates. In: Proceedings of the 2002 ACM symposium on Applied computing. SAC '02, New York, NY, USA, ACM (2002) 1113–1120
23. van der Hoek, A.: Design-time product line architectures for any-time variability. Science of Computer Programming **53**(3) (2004) 285–304 Software Variability Management.
24. Bragança, R., Machado, R.J.: Run-time variability issues in software product lines. In: In: ICSR8 Workshop on Implementation of Software Product Lines and Reusable Components. (2004)
25. Goedicke, M., Pohl, K., Zdun, U.: Domain-specific runtime variability in product line architectures. In Bellahsène, Z., Patel, D., Rolland, C., eds.: Object-Oriented Information Systems. Volume 2425 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2002) 384–396
26. Cetina, C., Fons, J., Pelechano, V.: Applying software product lines to build autonomic pervasive systems. In: Software Product Line Conference, 2008. SPLC '08. 12th International. (2008) 117–126