The Journal of Systems and Software 112 (2016) 78–95

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# An automatic process for weaving functional quality attributes using a software product line approach

Jose-Miguel Horcas*, Mónica Pinto, Lidia Fuentes

*CAOSD Group, Universidad de Málaga, Andalucía Tech, Spain*

ARTICLE INFO

*Article history:*
Received 7 May 2015
Revised 28 September 2015
Accepted 3 November 2015

*Keywords:*
Quality attributes
Software product lines
Weaving

ABSTRACT

Some quality attributes can be modelled using software components, and are normally known as Functional Quality Attributes (FQAs). Applications may require different FQAs, and each FQA (e.g., security) can be composed of many concerns (e.g., access control or authentication). They normally have dependencies between them and crosscut the system architecture. The goal of the work presented here is to provide the means for software architects to focus only on application functionality, without having to worry about FQAs. The idea is to model FQAs separately from application functionality following a Software Product Line (SPL) approach. By combining SPL and aspect-oriented mechanisms, we will define a generic process to model and automatically inject FQAs into the application without breaking the base architecture. We will provide and compare two implementations of our generic approach using different variability and architecture description languages: (i) feature models and an aspect-oriented architecture description language; and (ii) the Common Variability Language (CVL) and a MOF-compliant language (e.g., UML). We also discuss the benefits and limitations of our approach. Modelling FQAs separately from the base application has many advantages (e.g., reusability, less coupled components, high cohesive architectures).

## 1. Introduction

The quality of a software system is measured by the extent to which it possesses a desired combination of quality attributes (QAs) (Barbacci, et al., 19950) such as usability, confidentiality, reliability, security or scalability. Some quality attributes (QAs) can be modelled using software components and are normally known as functional quality attributes (FQAs) (Juristo et al., 2007). Examples of FQAs are security (e.g., to allow access control), usability (e.g., to provide contextual help) or error handling (e.g., to respond to the occurrence of errors and exceptions). Note that other QAs (i.e., those related to non-functional requirements) such as cost, efficiency or portability cannot be directly mapped to functional software components, but they can be mapped to architectural or implementation decisions, so they are beyond the scope of this paper.

In order to satisfy application requirements, apart from core functional and non-functional requirements, the software architect should pay special attention to those that can be modelled as FQAs. FQAs are characterised by the following aspects: (1) they are recurrent — i.e., FQAs are normally required by several applications (e.g., security); (2) most FQAs crosscut the system architecture; and (3) they require the incorporation of specialised components inside the architecture (e.g., an authorisation mechanism to satisfy the security FQA). Normally, FQAs are modelled and tailored to a given application, with functional components that are part of the core architecture. But, modelling FQAs separately from the base application has many advantages (e.g., reusability improvement, less coupled architectures, etc.). For instance, an encryption algorithm used to encrypt the information to ensure confidentiality does not depend on the application that needs it.

Modelling FQAs is a complex task, firstly because they are usually composed of many concerns. The security FQA, for example, is composed of confidentiality, integrity, access control and authentication, among others. Secondly, different applications may require different levels of an FQA (e.g., different security levels). For example, a specific application may require access control, encryption, and anonymity while another may require only encryption, or may require a different kind of encryption algorithm. Thirdly, some of the concerns of an FQA may have dependencies between them. For example, the confidentiality concern depends on the encryption concern to ensure that all the information is encrypted and cannot be obtained by third persons. Furthermore, some FQAs affect each other, so dependency relationships between different FQAs must also be considered. For instance, the contextual help concern of the usability FQA depends on the authentication concern of the security FQA to be able to offer

* Corresponding author. Tel.: +34625257121.
*E-mail addresses:* horcas@lcc.uma.es, migueli_jordan@hotmail.com (J.-M. Horcas), pinto@lcc.uma.es (M. Pinto), lff@lcc.uma.es (L. Fuentes).

http://dx.doi.org/10.1016/j.jss.2015.11.005
0164-1212/© 2015 Elsevier Inc. All rights reserved.

customised help based on the user's previous experience with a given application.

To summarise, there is much variability in FQAs and different dependency levels. Therefore, specifying the set of FQA components and connections that fulfil the application requirements is not a trivial task for the software architect. Our goal is to alleviate the software architect's tasks with respect to FQAs by: (i) defining a family of FQAs with commonalities, variabilities and dependencies; and (ii) implementing a process able to automatically generate the final application architecture that includes the customised FQAs.

The variability of FQAs can be modelled by using different techniques provided by traditional Software Product Line (SPL) (Pohl et al., 2005) approaches. Reviewing the literature, the conclusion can be drawn that little care is taken to model variability of the functional part of QAs (Etxeberria et al., 2008), and normally the focus is on modelling the functional variability of the application. Some approaches propose techniques for analysing and/or reasoning about the impact of functional variants on the quality of applications derived from an SPL, principally concentrating on non-functional QAs such as performance, availability, cost, or latency (Benavides et al., 2005; González-Baixauli et al., 2004; Sinnema et al., 2006). Others address FQAs variability (e.g., QADA Matinlassi et al., 2002, RiPLE-DE Cavalcanti et al., 2011), but they model these FQAs intermingled with the functional components, as part of the domain analysis of an SPL, and not separately as we propose. The main drawback of these latter approaches is that they do not provide means to easily reuse FQAs and their dependencies across several applications, nor do they facilitate the customisation of FQAs to each individual application.

In order to define a family of FQAs following a generic SPL approach, we need a language to specify and model the variability of FQAs. According to Czarnecki et al. (2012) Feature Models (FMs) are the most used variability language, which model variability by means of high-level features that are close to requirements specification. More recently, the Common Variability Language (CVL Haugen et al., 2012) was proposed as a standard. Both alternatives are currently well accepted by the SPL community, and can be used in our approach.

Independently of the variability language used, once an architectural configuration of the FQAs has been generated, a process to incorporate it into the architecture of the base application non-intrusively, is required. For weaving FQAs with the base application we will use some aspect-oriented mechanisms. By combining SPL and aspect-oriented software development (AOSD) technologies, we have defined a generic process to: (i) specify and model the variability and dependencies among FQAs, defining a reusable family of FQAs; (ii) customise the FQAs to fulfil the application requirements and automatically generate an architectural configuration of FQAs; and (iii) weave the customised FQAs into the software architecture of the base application without manually modifying it. We present and compare two instantiations of our generic process using different variability languages and architecture description languages: (1) with feature models and AO-ADL, an aspect-oriented architecture description language (Lence et al., 2011; Horcas et al., 2013); and (2) using the Common Variability Language (CVL) and a MOF-compliant language such as UML (Horcas et al., 2014). We illustrate our approach with an e-voting case study and quantitatively evaluate both approaches by using suitable metrics (degree of dependency, variability, automation, separation of concerns) to assess the benefits of each approach. Also, we discuss the benefits and disadvantages of both implementations of our approach.

The remainder of the paper is organised as follows: Section 2 presents the challenges addresed in this work and motivates it with a case study. Section 3 overviews our approach. In Section 4 we explain in detail how we model FQAs with two different instantiations of our approach. The customisation and incorporation of the FQAs into the base application of our case study is explained in Sections 5 and 6, respectively. In Section 7 we evaluate our approach, while in Section 8 we identify and discusses the benefits and limitations of our approach. Section 9 discusses the related work. Finally, Section 10 concludes the paper.

## 2. Motivation and challenges

In this section we present the specific challenges addressed in this work and the motivating case study we use to illustrate our approach.

### 2.1. Challenges

In this section we describe the specific challenges addressed in our work related to FQA modelling and the weaving of a tailored FQA configuration into different applications.

**Challenge 1. Manage the variability of FQAs and their customisation according to the application requirements.** The issue of the high degree of variability in FQAs has been neglected or even ignored by most software architects as attention has mainly focused on functional variability (Cavalcanti et al., 2011). The challenge is to model all the possible FQA variation points independently of the final application, which is not a trivial task. *In this paper we define a family of FQAs, following an SPL approach, that supports the customisation of FQAs to satisfy the specific requirements of each application. We provide a process that configures FQA variation points in such a way that variable concerns that are not required by the base application are not incorporated into the final application. In this paper we explore the use of both FM and CVL languages to specify a family of reusable FQAs.*

**Challenge 2. Manage dependencies between FQAs.** In FQA modelling, dependencies between concerns that are part of the same FQA need to be taken into account, — i.e., *intraFQA-dependencies*. Furthermore, dependency relationships between different FQAs must also be considered, — i.e., *interFQA-dependencies*. These kinds of dependencies often go unnoticed by software architects, who are not domain experts in modelling all types of FQAs. *Using the support provided by the SPL approach these dependencies are automatically traced and incorporated into the solution even if they have not been explicitly selected by the software architect. For example, if a concern X depends on other concerns Y, W and Z, then these other concerns should be automatically incorporated into the solution even if they have not been explicitly selected by the software architect.*

**Challenge 3. Define architectural patterns with reusable FQAs.** Once the FQAs have been modelled as independently as possible, the customised FQAs need to be woven with the final application. The challenge here is to define a process that systematically integrates high-level quality solutions into the base architecture of a given application, but without having to either understand the inner working of the quality solutions, or break the application's core architecture, — i.e., architecture components should be completely unaware of the FQAs they are affected by. This is not a straightforward task since each FQA needs to be woven at different points of the base applications. Moreover, multiple views may be required to appropriately model the FQAs (e.g., behavioural view, structural view), and this makes the weaving process even more complex. *As part of our work we define different architectural weaving patterns, following the non-invasive weaving mechanism of aspect-orientation. For this we follow two different approaches: (i) use connector templates defined by the AO-ADL language; and (ii) use CVL and implement the corresponding model transformations.*

**Challenge 4. Support the approach with tools.** The approach presented in this paper is not viable without the required tool support. *In our approach we combine several tools for SPL and AOSD in order to completely automate the process of: (1) generating customised software architectures for the FQAs required by an application, and (2) weaving these software architectures with the software architecture of the core functionality of the application.*