Variability and Dependency Modeling of Quality Attributes

José Miguel Horcas, Mónica Pinto, Lidia Fuentes
CAOSD Group, University of Málaga, Spain
horcas@lcc.uma.es, pinto@lcc.uma.es, lff@lcc.uma.es

***Abstract*—Functional Quality Attributes (FQAs) are quality attributes that have strong functional implications and so can be easily modeled by software components. Thus, we use an aspect-oriented software product line approach, to model the commonalities and variabilities of FQAs from the early stages of the software development. However, FQAs cannot be modeled in isolation since they usually have dependencies and interactions between them. In this paper we focus on identifying and modeling the dependencies among different FQAs. These dependencies are automatically incorporated into the final software architecture of the system under development, even when the software architect may be unaware of them.**

***Keywords*- AO-ADL; dependencies; feature models; quality attributes.**

I. INTRODUCTION

The critical quality attributes (QAs) of a software system must be well understood and articulated early in the development of a system, so that the architect can design a software architecture that satisfies them [1]. Our work focuses only on those QAs that have strong functional implications and so can be easily modeled by software components (like security, usability, error handling, etc.), which we call functional quality attributes (FQAs).

Modeling FQAs is not a straightforward task due to several reasons. On the one hand, they are usually very complex, being composed by many concerns. For example, security is a FQA composed by authentication, access control, encryption and non-repudiation concerns, among others. On the other hand, the concerns that comprise a given FQA have dependencies and interactions between them. For example, the confidentiality concern will depend on the authentication, and encryption concerns. A FQA has also dependency relationships with other FQAs, since some concerns are shared and required by different FQAs. Security is a typical example, where their concerns are required to satisfy other FQAs (e.g. usability, adaptability).

Another difficulty of modeling FQAs is that each software system requires a variable number of FQA's concerns to be incorporated into its software architecture. For instance, one application may require the authentication and encryption concerns of security while another application may require the non-repudiation and the privacy security concerns. The concerns that are not required should be easily removable from the final software architecture.

Combining the use of SPLs and Aspect-Oriented Software Development (AOSD)[1] technologies, in [2] we proposed an Aspect-Oriented Software Product Line (AO SPL) approach to model the commonalities and variabilities of complex FQAs. Using this approach the software architect can automatically derive a valid configuration of a FQA that includes only the software components modeling those concerns that are required by a particular software system. However, in that work, we focused only on modeling one single FQA without considering that FQAs have dependencies and interactions between them that make it impossible to model them in insolation, being the main shortcomings in that case the following ones: (1) FQAs would include concerns that do not pertain to that FQA – e.g. authentication would be modeled as a concern of the usability FQAs because the authentication of the user is needed in order to provide him/her contextual help, and (2) the same concerns would be scattered in the models of several FQAs — e.g. the authentication concern would be modeled as a concern of both security and usability.

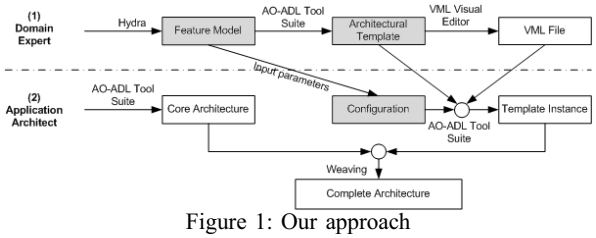

Figure 1: Our approach

The approach followed in this paper to avoid the aforementioned shortcomings consists on the identification of the dependencies between FQAs from the early stages of the development. Concretely, we extend our previous work to include the specification of the dependencies between the concerns of different FQAs. Thus, in our approach, experts on the domain of each FQA jointly specify the commonalities and variabilities of a set of FQAs, as well as all the dependencies among them. Then, the application architect selects the concerns that are required by the application according to the requirements specification document. This selection implies the automatic incorporation to the software architecture of any other concern of the same or other FQAs the selected concerns depend on, even if the software architect was unaware of these dependencies.

After this introduction, Section II describes the background, motivation, and a case study. In Section III we model

[1] http://www.aosd.net/

the dependencies between different FQAs. Section IV uses our case study to illustrate the customization of the FQAs. Finally, Section V discusses the related work and Section VI presents our conclusions and future work.

II. Background and Motivation

A. Background information

Our AO SPL approach is presented in Figure 1. In the upper half we see how an expert in the domain of the FQA specifies it. Following a SPL approach, we: (1) specify the feature model (FM) of the FQA (using the Hydra FM [3] specification tool), (2) define a template of the FQA software architecture (using the AO-ADL Tool Suite [4] for the specification of AO architectures [5]), and (3) link the features in the FM with the elements in the AO architecture (using the Variability Modeling Language (VML) [6])[2]. This is done only once for each FQA.

Then, for each application, the application architect creates a product configuration by selecting the features of the FQA that are required by that particular application. Using that configuration, as well as the AO-ADL template and VML file previously created, a template instance which includes only those components that model the selected FQA's concerns is automatically generated and woven with the core architecture of the application in an AO fashion.[3]

B. Case Study and Motivation

In this subsection we motivate our extension to the AO SPL approach previously presented using a case study. The case study consists of an application which provides information in real time on different themes such as sports scores, news, TV listings, etc. based on the interests of the user. The user decides what information about which themes, and the application notifies the user of events such as a goal of his favorite team or news about his interests.

In spite of the above core functionality, the following extra-functional requirements must also be considered:

1. All the information must be *encrypted* in order to avoid that third people can change it with fake information or can discover the preferences of the user.
2. *Contextual help* must be provided for the user based on the previous experience.
3. *Feedback* information is provided for the user to suggest other possible information that may be of his/her interest.

From the aforementioned requirements, the concern *encryption* (concern of the security FQA), and the concerns *feedback* and *contextual help* (concerns of the usability FQA) need to be included as part of the specification of the application software architecture.

In order to incorporate the FQAs, the application architect will select *encryption* from the security FM and *feedback* and *contextual help* from the usability FM. However, since in most cases the software architect will not be an expert in the security or usability domains, he/she may not be aware of all the dependency relationships between those FQAs. For instance, to provide contextual help to the user, he/she needs first to be authenticated to get customized help based on the previous experience of the user with the application. But the software architect may not identify the *authentication* concern as a requirement of the system because it was not explicitly included in the system requirements specification. So, the approach presented in [2], in which each FQA was modeled independently, is not enough to include these dependencies between concerns of different FQAs.

In order to avoid this situation, the most relevant contribution of our extension is that it allows the identification and modeling of dependencies between FQAs that are not always explicitly specified as part of the application requirements. Moreover, these dependencies are automatically incorporated to the instantiated software architecture with no additional effort required by the application architect.

III. Dependency Modeling of FQAs

In modeling FQAs, we distinguish two kinds of dependencies: (1) **intraFQA-dependencies:** the dependency relationships between the concerns of a FQA. For example, the *encryption* concern of the security FQA requires a *key storage*; and (2) **interFQA-dependencies:** the dependency relationships between concerns belonging to different FQAs. For example, a *database storage* with a secure access way (concern of the persistence FQA) requires *access control* (concern of the security FQA).

In a FM, these dependencies can be represented either graphically or textually. Using these two mechanisms it is possible to represent the following constraints:

- **tree constraints:** they represent dependencies between concerns that belong to the same branch of the tree. Some IntraFQA-dependencies can be modeled in this way.
- **cross-tree constraints:** they represent dependencies between concerns that are in different branches of the tree. InterFQA-dependencies are always modeled in this way because the concerns of different FQAs will always be in different branches of the tree. Also, some intraFQA-dependencies need to be modeled in this way.

The following subsection focuses on how we model interFQA-dependencies which are the most interesting and difficult to identify.

A. Dependencies between different FQAs

In this section we go through the steps of our approach in which the modeling of the dependencies is specially relevant (steps in a dark color in Figure 1).

Feature Model. In order to model the interFQA-dependencies we model all the FQAs that have dependency relationships between them together in the same FM. Thus,

[2]These tools are described in [2] and can be downloaded from [4], [3].

[3]Core architecture: software architecture modeling the core functionality of the application, without including the FQAs.

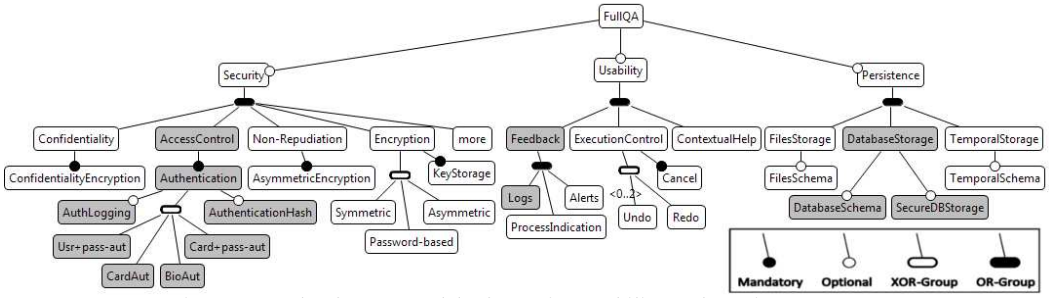

Figure 2: Hydra feature model of security, usability and persistence FQAs.

```
1. (AuthenticationHash or IntegrityHash) implies Hash;
2. AsymmetricEncryption implies Asymmetric;
3. ConfidentialityEncryption implies Encryption;
4. ContextualHelp implies Authentication;
5. AuthLogging implies Logs;
6. (Usr+pass-aut or CardAut or BioAut or Card-pass-aut) implies DatabaseStorage;
7. SecureDBStorage implies AccessControl;
```

Figure 3: Textual constraints of the feature model.

Figure 2 models security, usability and persistence in the same FM.[4]

In this joint FM, the interFQA-dependencies are textually specified using cross-tree constraints (see constraints 4-7 in Figure 3). For instance, constraint 4 indicates that the selection of the `ContextualHelp` feature of the usability FQA will imply selecting also the `Authentication` feature of the security FQA (`ContextualHelp` implies `Authentication`). Moreover, since the `AccessControl` feature is the parent feature of `Authentication`, access control is also automatically incorporated inside the resulting product configuration. Finally, the software architect must select between the optional features that are children of authentication – e.g. the use or not of a logging (`AuthLogging`) and hash (`AuthenticationHash`), or the way of authentication.

However, in our example, the features that are automatically incorporated into the product configuration once the `ContextualHelp` feature is selected, are not only limited to the ones mentioned above. Instead, let us suppose that once `Authentication` has been incorporated, the application architect selects the optional `AuthLogging` feature. In this case, according to constraint 5 in Figure 3 (`AuthLogging` implies `Logs`), the `Logs` feature of the usability FQA is also selected, as well as the `Feedback` feature, which is the parent feature of `Logs`. Moreover, according to constraint 6, the `DatabaseStorage` feature of the persistence FQA is incorporated whether the application architect chooses to use either a user-password, a card, a biometric or a card-password authentication mechanism. Finally, adding the `DatabaseStorage` feature forces the application architect to optionally choose the `SecureDBStorage` feature and, if chosen, this feature implies the `AccessControl` feature of the security FQA. To clearly show the consequences of selecting the `ContextualHelp` feature in our approach with interFQA-dependencies, in Figure 2 we show the branches of the tree that are involved in a dark color.

Architectural Template. The AO-ADL architectural template in the new version of our approach is extended to contain the definition of the features of all the FQAs, as well as the relationships between them. In order to do that we define the architectures with two levels of granularity. In the first level, shown on the left of Figure 4, there is a composite component representing each FQA. Thus, `Security`, `Usability` and `Persistence` in Figure 4 are composite parameterized components that represent three sub-templates for modeling security, usability and persistence, as well as their interFQA-dependencies. The right of Figure 4 shows the security and the persistence templates.

[4]Neither all the FQAs, nor all the concerns of each FQA have been included.

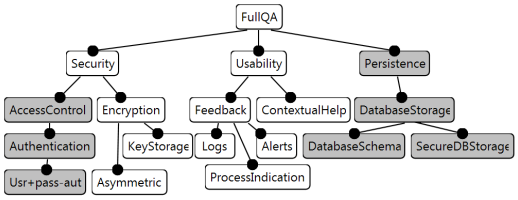

Figure 5: FQAs configuration.

The VML file. The VML file specifies the matching between the features chosen from the FM and the actions to be applied on the architecture. This file and its description can be found in [2].

IV. Creating a configuration of the FQAs

At this point, the application architect has a FM and an architectural template that model all the FQAs that are needed, and the dependencies between them.

Firstly, a valid configuration is created depending on the FQAs requirements of the application. In order to do that, we use the Hydra tool and the FM previously generated. Figure 5 shows a valid configuration that satisfies our case study extra-functional requirements. Nodes in a dark color represent the features that are not explicit requirements of the application, and that have been automatically added to the configuration due to the interFQA-dependencies that were modeled in section III-A. Hydra automatically selects all the necessary features to satisfy both the intraFQA- and interFQA-dependencies.

The next step is to instantiate the FQAs template with the chosen configuration. Figure 6 shows the architecture obtained for the selected FQAs. Unrequired architectural elements have been automatically removed from the templates, and others such as Authentication or Encryption, which were parameters, have been instantiated.

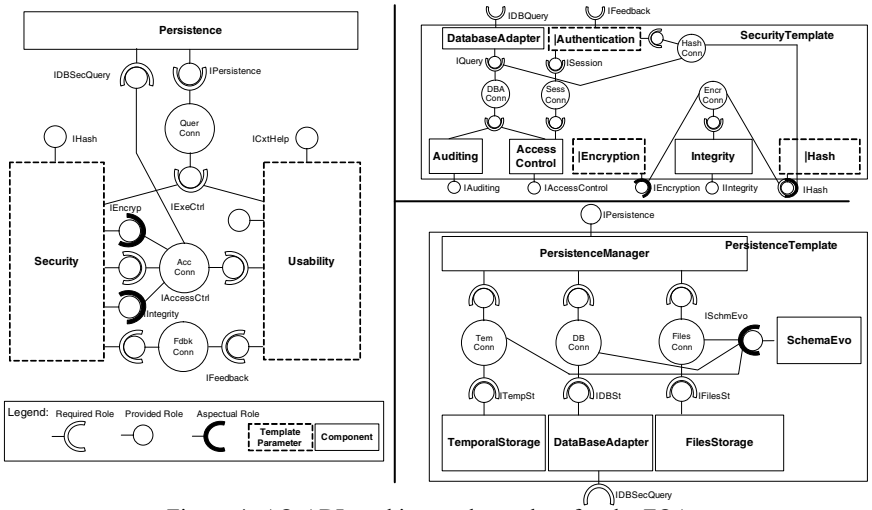

Figure 4: AO-ADL architectural template for the FQAs.

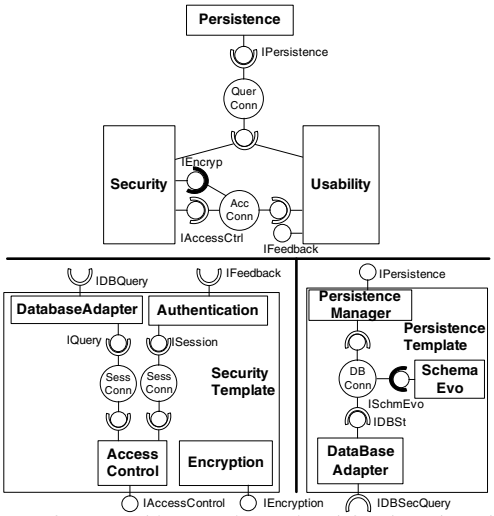

Figure 6: Software architecture instantiated for the selected FQAs.

V. Related Work

In [7] the authors decompose an overall diagram into a set of individual FMs, and propose a matrix-based approach to maintain and managing the information about feature dependencies between different FM trees. However, from the point of view of the domain experts, our approach facilitates the modeling of the QAs in the same model specifying the dependencies in an explicit way, without the necessity of encoding them in an auxiliary structure as a matrix. Moreover, the main difference is that they do not focus on modeling FQAs, as we do. The support for modeling dependencies at the FM level already exist and thus our work focuses on using this existing support to improve and automate the modeling of FQAs and their dependencies from early stages of the development.

COVAMOF [8] is a framework that captures variability of QAs in terms of variation points and dependencies by using associations. Dependencies specify properties to the FMs that define values of the QAs such as performance or memory usage. In contrast to our proposal, this approach also address the variation of non-functional QAs.

VI. Conclusions and Future Work

In this paper, we have presented an extension to our AO SPL approach to model the dependencies between the concerns of different FQAs. The main contribution of this work is that our extension allows the identification of dependencies between FQAs that are not always explicitly specified as part of the application requirements. In addition, this also allows the automatic incorporation of these dependencies to the software architecture in a transparent way, from the point of view of the application architect. A goal achieved with our extension is that it avoids the duplication of shared concerns in different FQAs.

As part of our ongoing work, we are now exploring the possibility of defining a complete repository of FQAs ready to be customized and incorporated inside any application.

Acknowledgment

Work supported by the European INTER-TRUST 317731 and the Spanish TIN2012-34840 and P09-TIC-5231 projects.

References

[1] F. Bachmann and et al., "Designing software architectures to achieve quality attribute requirements," *IEE Proceedings*, vol. 152, no. 4, pp. 153–165, August 2005.

[2] R. Lence, L. Fuentes, and M. Pinto, "Quality attributes and variability in ao-adl software architectures," in *ECSA*. pp. 7:1–7:10. 2011

[3] CAOSD Group University of Málaga, "Hydra project website," http://caosd.lcc.uma.es/spl/hydra/.

[4] CAOSD Group University of Málaga, "AO-ADL project website," http://caosd.lcc.uma.es/aoadl/.

[5] M. Pinto, L. Fuentes, and J. María Troya, "Specifying aspect-oriented architectures in AO-ADL," *Information & Software Technology*, vol. 53, no. 11, pp. 1165–1182, 2011.

[6] S. Zschaler, "Vml*: A generative infrastructure for variability management languages," 2009.

[7] H. Ye and H. Liu, "Approach to modelling feature variability and dependencies in software product lines," *Software, IEE Proceedings*, vol. 152, no. 3, pp. 101–109, june 2005.

[8] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Modeling dependencies in product families with covamof," in *Engineering of Computer Based Systems, 2006*, 2006, pp. 9 pp.–307.