WeaFQAs: A Software Product Line Approach for Customizing and Weaving Efficient Functional Quality Attributes

UNIVERSIDAD DE MÁLAGA

José Miguel Horcas Aguilera

Departamento de Lenguajes y Ciencias de la Computación

Universidad de Málaga

Supervised by

Prof. Dr. Lidia Fuentes Fernández

Dr. Mónica Pinto Alarcón

July 2018

UNIVERSIDAD DE MÁLAGA

AUTOR: José Miguel Horcas Aguilera

http://orcid.org/0000-0002-7771-0575

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

UNIVERSIDAD DE MÁLAGA
DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

La Prof. Dr. Doña Lidia Fuentes Fernández, Catedrática de Universidad, perteneciente al Área de Ingeniería Telemática, y la Dr. Doña Mónica Pinto Alarcón, Titular de Universidad, perteneciente al Área de Lenguajes y Sistemas Informáticos, de la E.T.S. de Ingeniería Informática de la Universidad de Málaga,

certifican que Don José Miguel Horcas Aguilera, Ingeniero en Informática, ha realizado en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, bajo nuestra dirección, el trabajo de investigación correspondiente a su Tesis Doctoral titulado:

WeaFQAs:
A Software Product Line Approach for Customizing and Weaving Efficient Functional Quality Attributes

Revisado el presente trabajo, estimamos que puede ser presentado al tribunal que ha de juzgarlo, y autorizamos la presentación de esta Tesis Doctoral en la Universidad de Málaga.

Málaga, Julio de 2018

Fdo.: Lidia Fuentes Fernández	Fdo.: Mónica Pinto Alarcón
Catedrática de Universidad	Titular de Universidad
Área de Ingeniería Telemática	Área de Lenguajes y Sistemas Informáticos

Acknowledgements

This thesis has been supported by the European project INTER-TRUST FP7-317731; the Spanish projects MAGIC P12-TIC1814, HADAS TIN2015-64841-R, MAVI TIN2012-34840, and FamiWare P09-TIC-5231; and the University of Málaga.

Special Acknowledgements

Is it too late now to say 'thanks'?

♪$_1$ ♪$_2$ ♪$_3$ ♪ ♪$_5$ ♪$_6$ ♪$_7$ ♪

Es un dilema complicado ponerse a escribir esto ahora que ya parece que ha acabado el largo proceso, pero toca reconocer la parte que les toca a los que estuvieron conmigo en el camino.

A mis supervisoras, o como yo las llamo fuera, las jefas. Habrá más tesis, pero sólo una es la primera, ¿será esta?

A los compañeros del laboratorio 333, los que siguen y los que se fueron. Mención especial a las sobremesas de los viernes que no puedo "`negar`" que me encantan.

A los amigos, que cada vez son más y mejores, los de allí y los de aquí, los del día y los de la noche, aunque sólo duren una canción. Y en especial a mis petard@s favorit@s.

A la familia, que siempre está ahí y ha contribuido a que esto sea posible, aunque nunca vayan a leer este documento.

Y a _____ que, o bien me he olvidado de darle las gracias explícitamente, o bien no conozco todavía pero se tendrá que leer esta tesis algún día; y será *Lo que más.*

Contents

List of Figures

List of Tables

Part I

Thesis

Chapter 1

Introduction

A *quality attribute* (QA) is a characteristic that affects the quality of software systems [2]. The quality of a software system is the degree to which software possesses a desired combination of attributes such as usability, security, performance, energy efficiency, persistence, confidentiality, reliability, and scalability. The list of documented QAs is very long and they can affect the design, run-time behavior, and user experience of a software system. Quality attributes must be well understood and articulated from the early stages of the development process. Indeed, QAs play a critical role in the architecture elicitation phase, serving as selection criteria to choose from among a great number of alternative designs and ultimate implementations.

All but the most trivial application have *non-functional requirements* (NFRs) that can be expressed in terms of quality attribute requirements [3]. In some domains the fulfillment of QAs is even more critical that addressing functional requirements. For example, in embedded systems time or safety issues may be of great importance and their modeling has even bigger complexity than functional requirements modeling. Even so, some QAs actually describe behavioral properties (e.g., security, usability) and should be treated the same way as functional requirements [4]. For instance, to satisfy the security QA, applications must include an encryption component that modifies the system behavior to provide confidentiality. The association of a function (e.g., the encryption of a message) to a goal (e.g., providing security) is known as the *operationalization* of the QA, in the

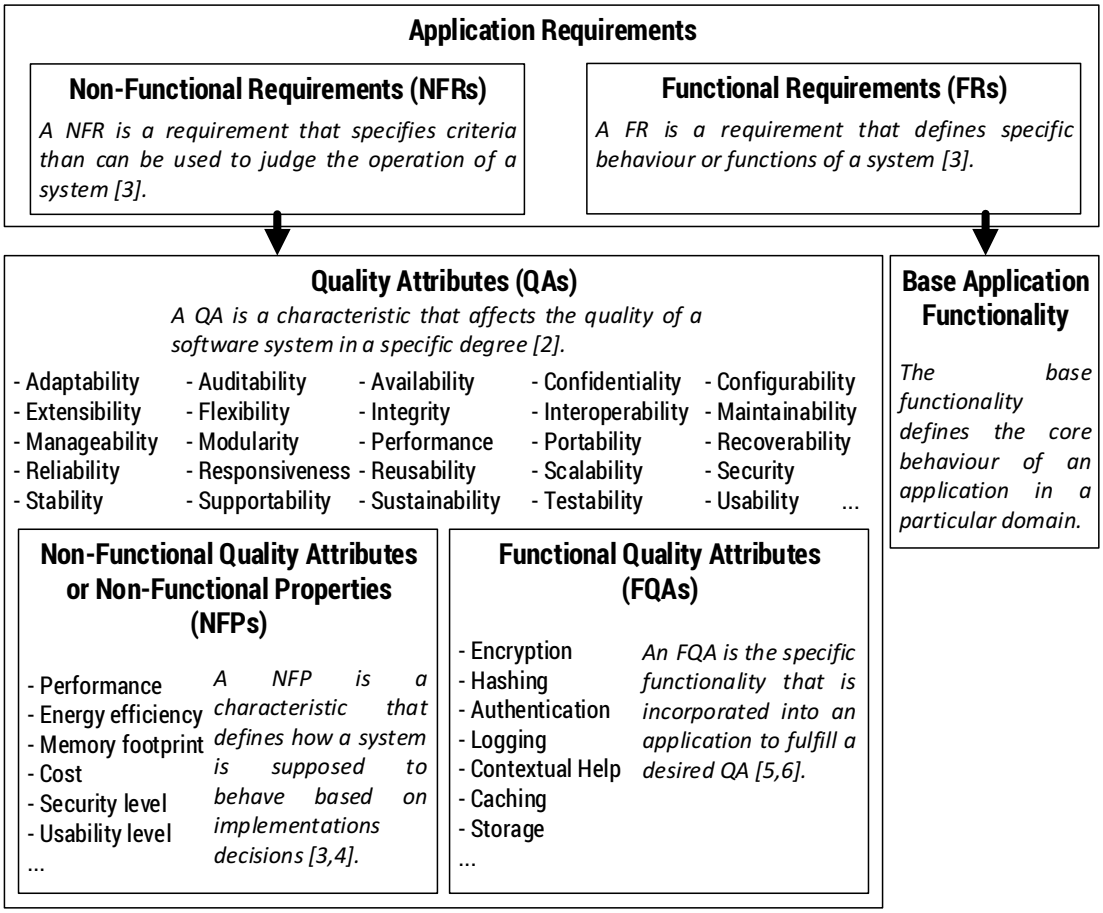

Figure 1.1: Definitions and classification of the terminology used in this thesis for quality attributes.

sense that the function specifies how a goal can be made operational [5]. This thesis introduces the concept of *Functional Quality Attribute* (FQA) as the specific functionality that is incorporated into the application to fulfill the desired QAs, as it describes the functional behavior needed to satisfy specific QAs [6]. For FQAs the operationalization is encapsulated in a distinct set of additional components. Examples of FQAs are encryption to satisfy confidentiality, hashing to satisfy integrity, authentication to satisfy access control, caching to improve performance, storage to satisfy persistence, and logging and contextual help to satisfy usability. In contrast, in this thesis, we use the term *non-functional QAs* or *non-functional properties* (NFPs) for those QAs related to non-functional requirements

that cannot be directly mapped to functional software components, but they can be mapped to architectural or implementation decisions. Examples of NFPs are performance, energy efficiency, memory footprint, and cost. Figure 1.1 clarifies the terminology used in this thesis. Note that QAs can be composed by many other concerns or QAs. Security, for example, is composed of confidentiality, integrity, access control, and authentication, among others concerns.

The kind of functionality described by the FQAs has some particular properties, which differentiate it from the base functionality of the application. First, FQAs are recurrent that means they are normally required and can be reused in several different applications. Despite this, each application may require a different variant or configuration of the FQA. While one application may require the authentication and encryption FQAs to satisfy the security concerns, another application may need the non-repudiation or hashing FQAs to satisfy the security concerns. Moreover, each FQA is also composed by many different functionality concerns (e.g., algorithms, implementation parameters, etc.). For example, to incorporate encryption in an application, a particular encryption algorithm, key length, block cipher mode of operation, and padding need to be selected.

Second, FQAs have dependencies relationships between them. A *dependency* (or interaction) is a relationship in which one FQA's concern uses or depends on another FQA's concern to satisfy a QA requirement. Dependencies can be found between concerns of the same FQA (called *intraFQA-dependencies*) or between concerns of different FQAs (called *interFQA-dependencies*). An example of an intraFQA-dependency is confidentiality that depends on encryption to ensure that all the information is encrypted and cannot be obtained by third persons; or a particular block cipher mode of operation that can only be used with specific encryption algorithms. An example of an interFQA-dependency is the contextual help FQA that to satisfy the usability QA requires the authentication FQA from security to be able to offer customized help based on the user's previous experience with a given application.

To summarize, there is much variability in FQAs and different dependency levels, and thus, modeling FQAs is a very complex task. Certainly, modeling FQAs can be seen as modeling a "family of products" in the sense used in Software Product Line (SPL) approaches [7]. In the context of FQAs, a "product configu-

ration" is the subset of FQAs' concerns that satisfy the quality requirements of a particular application (e.g., security, persistence, error handling). In addition, the components of the FQAs that are required into the application to satisfy the QAs often affect other non-functional QAs (NFPs) such as the performance and energy efficiency of the system. These NFPs usually compete and conflict with each other. For example, using the greenest implementation of a functional component reduces the energy consumption, but can penalize system performance. Therefore, if a system requires to optimize both energy efficiency and performance QAs, a compromised solution is needed. Indeed, the desired quality degree to accomplish in terms of NFPs (energy efficiency, performance) will, in turn, strongly influence the achievement of FQAs that describe behavioral properties (e.g., security, usability). It is very important to investigate the relationships between FQAs and NFPs jointly. Finding the optimum configuration of the FQAs that satisfies all the application's requirements and that additionally takes into account NFPs is a difficult and error-prone task to carry out manually. We define an *usage model* for each FQA (e.g., encryption) as part of the variability model of the FQAs. The usage model includes the set of variables and parameters of each FQA (e.g., the encryption algorithms, operation mode, key length) that can affect NFPs. This model allows application developers to analysis the relationships between FQAs and NFPs, by instantiating it with the variable values (e.g., size of the object to be encrypted) according to the part of the application where the FQA is required.

Another difficulty of modeling FQAs is that most FQAs are *crosscutting concerns*, the behavior of which is tangled and/or scattered with the base behavior of the applications being affected by them. For instance, security needs to be usually ensured in different points of an application. In particular, an FQA may entail the incorporation of several additional components in different places such as encryption that requires a place where to integrate the encrypt functionality to encrypt the data, and another place where to decrypt the same data — i.e., encryption is *scattered* among different components of the applications. Also, a component may need to be both secure and persistent, therefore security and persistence FQAs are *tangled* in the same component of the base application. One software technology broadly used to cope with crosscutting concerns is Aspect-Oriented Software

Development (AOSD)[1] [8], where crosscutting concerns are modeled in separated entities (i.e., the aspects) that are then "woven" (i.e., composed) with the components of an application in such a way that the core components are oblivious to them[2]. This means that the core components are not aware of where/how/when crosscutting concerns are incorporated. Using AOSD, FQAs can be developed separately from the applications, customized according to the specific application requirements, and incorporated into the application in a non-invasive way. This is possible because the definition of the FQAs are independent from the applications that required them (e.g., the implementation of an encryption algorithm is independent from the application that uses it). Modeling FQAs separately from the base application has many advantages: reusability, less coupled architectures, facilitating maintenance, and improving evolution. Finally, FQAs can evolve in the future. First, QAs requirements of the applications can change over time and this implies to update the previous configurations of FQAs deployed within the applications. Second, frameworks and libraries that provide implementations of FQAs are continuously evolving and updating their technologies in order to be competitive in the market: new authentication methods (e.g., social network identity authentication, biometrics) or novel persistence techniques (e.g., sharding databases, affinity groups) frequently appear. This implies that the "family of FQAs" needs to be upgraded with the new features, and thus, the deployed configurations of FQAs need to be updated accordingly.

To address all the aforementioned problems, in this thesis we propose WeaFQAs, an Aspect-Oriented Software Product Line (AO-SPL) approach to manage FQAs, including their modeling and customization separately from the applications, and the later incorporation into the applications by combining the benefits provided by several technologies: Software Product Lines (SPLs), Aspect-Oriented Software Development (AOSD) and Model-Driven Development (MDD).

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

[2] The core components of an application are those that provide the main functionality of the application.

Chapter 2

Background Information

This section presents the background that is necessary to understand the work presented in this thesis. Concretely, we introduce the main concepts of Software Product Lines (Section 2.1), including the existing techniques to model variability, with special emphasis on the Common Variability Language. We also describe the main technologies used in the thesis: Aspect-Oriented Software Development (Section 2.2), and Model-Driven Development (Section 2.3).

2.1 Software Product Lines (SPLs)

A *Software Product Line (SPL)* is "a set of software-intensive systems that share a common, managed set of *features* satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [7]. A feature is a characteristic or end-user-visible behavior of a software system. Features are used in SPL engineering to specify and documenting commonalities and differences of the products, and to guide structure, reuse, and variation across all phases of the software life cycle. A specific product is identified by a subset of features, called a *feature selection* or *configuration*. Since not all feature selections are valid and specify meaningful products, there are constraints on the feature selection, called *dependencies*, and these dependencies are modeled explicitly in SPLs.

The classical SPL engineering paradigm separates two processes (Figure 2.1):

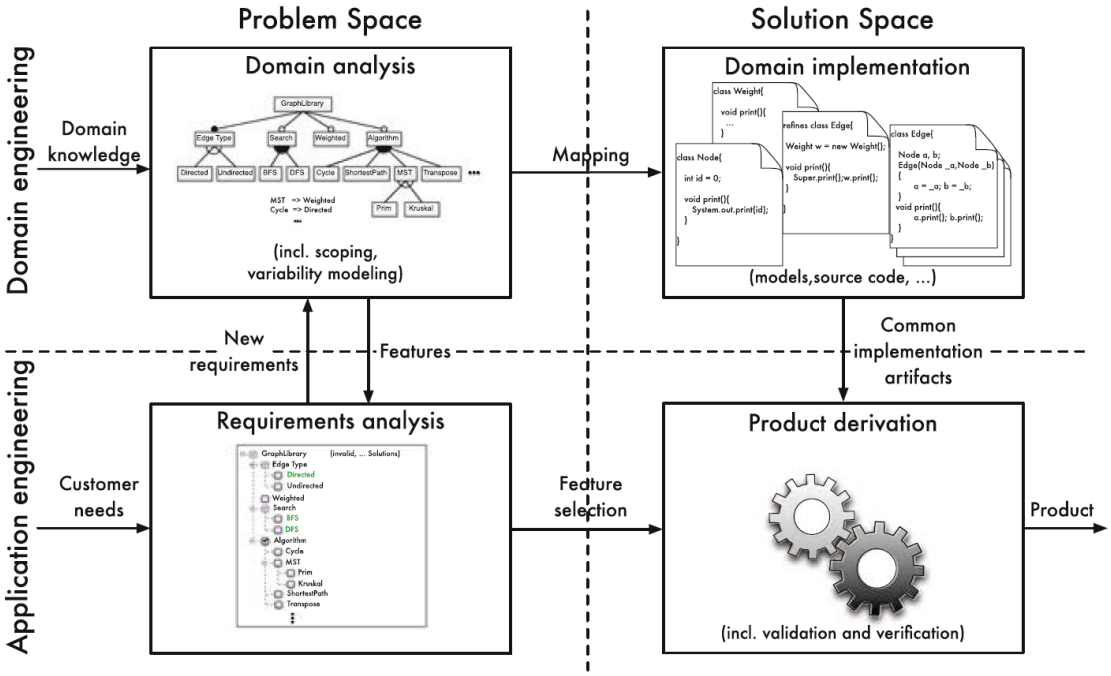

Figure 2.1: Overview of the SPL engineering process (taken from [1]).

Domain engineering, and *Application engineering* [7]. Domain engineering is the process of analyzing the domain (i.e., an area of knowledge) of an SPL and developing reusable artifacts. Application engineering is the process of developing a specific product for the needs of a particular customer. The specific characteristics of SPLs also lead to a separation between *problem space* and *solution space*. The problem space takes the perspective of stakeholders and their problems, requirements, and views of the entire domain and individual products. Features are domain abstractions that characterize the problem space. The solution space represents the developer's and vendor's perspectives. The solution space is characterized by the terminology of the developer and covers the design, implementation, validation and verification of features and their combinations in suitable ways to facilitate systematic reuse.

2.1.1 Variability Modeling

Variability modeling is the main activity of SPLs, where the commonalities and variabilities of the system are specified. Variability modeling is often realized with

three different approaches: (i) expressing the commonality and variability with feature models (Section 2.1.1.1); (ii) expressing the variability as annotations in the base model[1] (Section 2.1.1.2); and (iii) using a variability language like CVL (Section 2.1.1.3).

2.1.1.1 Feature Models

The most used variability model is the feature model [9] that has the advantage of a formal basis. It was first introduced by Kang as part of the Feature-Oriented Domain Analysis (FODA) in [9]. FODA is a domain analysis method that focuses on the "features" of the domain's systems, where a feature is perceived as an aspect or characteristic of a system that is visible to the end-user. FODA provides a tree-structural notation to model feature level commonality and variability graphically (i.e., feature diagrams). Features can be *optional* or *mandatory*, and can have group of features such as *or* groups or alternative groups (*xor*). Additional constraints between features may span large parts of the feature diagrams and are therefore called *cross-tree constraints*, and they can be represented either graphically or textually.

Several extensions of FODA have been proposed: Feature-Oriented Reuse Method (FORM) [10], FeatureRSEB [11], Generative Programming [12], PLUSS [13], Cardinality-based feature models [14, 15], and feature diagrams with UML multiplicities [16, 17], among others.

The main limitation of feature models is that they do not provide a mechanism to link the common and variable features with the realization artifacts (e.g., components of a software architecture, code, etc.), and so they must be supported by a separate variability language that makes that mapping — i.e., feature traceability.

2.1.1.2 Annotations

An alternative to express variability is expressing it as annotations in the base model. This means that annotated elements of the base model will become elements of specific products (or not) according to resolutions of related variability models. Annotations on the base model can be implemented as extensions to

[1] The model of the base application on which variability is defined.

the base modeling language such as the UML profiles with stereotypes [18, 19] or using C preprocessors [20] at the code level. The advantage of this approach is that model elements that are subject to variability are clearly marked, while the disadvantages are that annotations prevent the reuse of both the application core architecture and the variability model, and that the tool support for automatically generating all the valid architecture is a long way from being mature enough [1].

2.1.1.3 The Common Variability Language (CVL)

The Common Variability Language (CVL) [21] is a domain-independent language for modeling variability. Using a separate variability language like CVL allows us to represent variability in separated models with more than one set of variation points and resolutions rules expressed for each base model. Base models do not contain any variability information, which improves the reusability of both the base models and the variability models. Since in this thesis we use CVL extensively, in this section we detail the main characteristics of CVL and how it works.

CVL allows the specification and resolution of variability over models defined in any Meta-Object Facility (MOF)[1] compliant language. An overview of the CVL approach can be seen in Figure 2.2. CVL specifies, in separate models, the variability that can be applied to a base model. The *base model* is a model in the domain language that can be defined using a MOF-based metamodel and does not contain any information about variability. The variability information is separately specified in a *variability model*, according to the CVL metamodel. How the variability model can be resolved to produce a configured new model from the base model is described as a feature selections in the *resolution models* (or configuration models). CVL defines an executable engine to automatically (*materialize*) a *resolved model*. The materialization process is in charge of transforming a base model into a product model by applying the variation points. The resolved model is a fully configured product model with the variability resolved.

In CVL, the variability model consists of two main parts (Figure 2.3): (1) an abstract level with *variability specifications* (*VSpecs tree*), and (2) a concrete level with *variation points* (*VPs*). VSpecs are tree structures representing choices

[1] `http://www.omg.org/mof/`

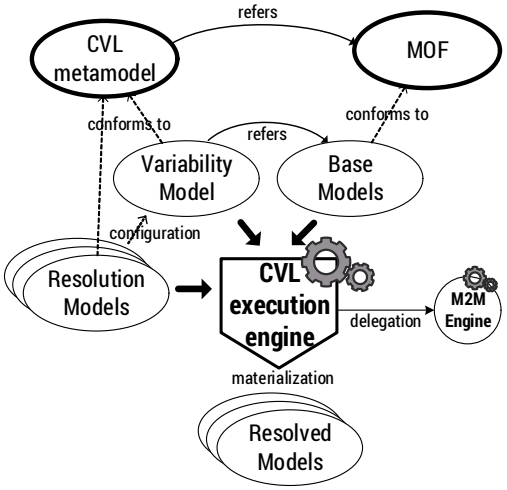

Figure 2.2: The CVL approach.

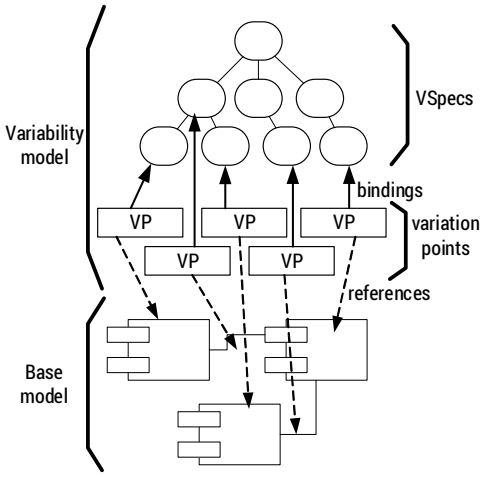

Figure 2.3: CVL variability model.

(features) and can include logical constraints defined in a subset of the Object Constraint Language (OCL). There are different types of VSpec: *Choices* (yes/no decision), *Variables* (attributes), *VClassifiers* (clonable features), and *Composite VSpecs* (modularization units). Variation points define the points of the base model that are variable and can be modified during the materialization process. Variation points also specify how the elements of the base model are modified by defining specific modifications to be applied by means of model-to-model (M2M) transformations. The semantics of these transformations is specific to the kind of variation point. Some of the variation points supported by CVL are the existence of elements of the base model (*ObjectExistence*) or the links between them (*LinkExistence*), the assignment of an attribute's value (*ParametricSlotAssignment*), or the replacement of a set of elements with another set of elements (*FragmentSubstitution*). An important type of variation point is the *Opaque Variation Point* (*OVP*) that enables defining new custom model transformations that are not pre-defined in CVL[1]. During CVL's execution, the CVL engine delegates its control to an M2M engine in charge of executing the transformations defined by the variation points (as shown in Figure 2.2).

CVL is more suitable for use at the architectural level, since it defines links between the variability specification and the product line architecture through the

[1]The complete taxonomy of variation points of CVL is available in [22].

CVL variation points. The advantages of using CVL are: (i) it is an MOF-based variability language and this means that any MOF-based architectural language can be used with the variability information of CVL; (ii) the links between the variability and base architectural models make it possible to automatically generate software architecture configurations, ensuring that they fulfill the variability specification, (iii) the semantic of the CVL variation points can be extended (e.g., using the OVP variation point), and (iv) CVL includes the most important characteristics of similar variability models (e.g. feature models) such as cardinality of variation points, cross-tree constraints, etc. Due to all these advantages there is a great interest in the SPL community in adopting CVL in their proposals [23, 24]. But, since both the CVL language and its tool support are novel, the effort of using CVL is currently considerable.

2.2 Aspect-Oriented Software Development (AOSD)

Aspect-Oriented Software Development (AOSD)[1] [8] aims to achieve separation of *crosscutting concerns*. A crosscutting is a structural relationship between the representation of two concerns. AOSD copes the limitation of the traditional software technologies (e.g., Object-Oriented Programming, Component-Based Software Development) to appropriately modularize crosscutting concerns as aspects.

In this way, FQAs are usually extra functional requirements of a system and can be seen as crosscutting concerns since they are normally tangled and/or scattered with the core behavior of the application (e.g., security). Incorporating these kinds of properties directly into the base code within the main functionality of an application causes *scattered code* —- i.e., the specification of a property or functionality is dispersed in more than one module —- and *tangled code* —- i.e., a module contains descriptions related with several functional and non-functional properties, which are intermingled with the base functionality of the module. Crosscutting concerns appear due to the fact that existing modeling techniques for composition and decomposition of a system support a "dominant dimension" that guides the

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

modeling process, implicitly or explicitly, through a certain hierarchical view of the organization of the system. This means that using hierarchical structures, an application can be decomposed in only one way at a time, called the dominant dimension or decomposition. All concerns that do not align with the dominant decomposition end up in scattered and tangled code. For instance, Object-Oriented Programming supports decomposition based on the dimension of the object: the class models a type of object and all the functionality associated with an object class (e.g., logging, trace) is described within the class. The disadvantage of these partitions is that many of the required functionalities of a system (in particular, the non-functional properties) are not well adapted to this decomposition criteria and have to be mixed with the concerns of the dominant dimension, affecting the separation of concerns. This problem is known as the *Tyranny of the Dominant Decomposition* [25].

AOSD promotes the principle of separation of concerns throughout all the phases of the software life cycle, including architecture design (Aspect-Oriented Architectural Design, AOAD). Since crosscutting concerns are normally hard to find, understand and work with, separating and specifying them at the architectural level enhances the evolution management of the system. AO architectures improve separation of concerns and component cohesion, which allows making correct and consistent changes to the concerns (FQAs in this thesis) of the system (update/add/remove functionality) before moving to implementation.

Next sections present the Aspect-Oriented Programming paradigm and the AO-ADL language to deal with crosscutting concerns at the code level and the architectural level, respectively.

2.2.1 Aspect-Oriented Programming (AOP)

Several techniques (e.g., design patterns, mixing classes) have been developed for dealing with the problem of modularization of crosscutting concerns. One of the most advanced and sophisticated technique is *Aspect-Oriented Programming* (AOP) [26]. AOP is a programming paradigm that improves modularization by allowing the separation of crosscutting concerns out of the dominant dimension of the programming language. In AOP, implementation of crosscutting concerns

are encapsulated in a new entity named *aspect*, and the code of the base application contains only the main functionality of the system excluding any reference to extra-functional properties. Interactions between base code and aspects are expressed through the *weaving rules*. To combine or weave the base applications and aspects, the base language and the aspects language must interact together in order to add the additional behavior contained in the aspects. Aspects are often described in terms of *join points*, *pointcuts* and *advices*. AOP languages must provide a joint point model that defines the points of the program execution which can be affected by the aspect (e.g., method call, function execution, field access,...). These points are called join points and a mean to refer to these join points is through the pointcuts. A pointcut is an expression that describes a set of join points where the code of the aspect should be executed in addition to the main behavior of the program —– i.e., the weaving rules. The additional code that affects the base program at the selected join points is the advice. Code that is added to the base program can be within aspect implementation or outside, depending on the language. In asymmetric approaches this code (i.e., the advice) is inside the aspect, but in symmetric ones, the advice is outside, being implemented as a normal class or module of the base programming language.

Modeling of crosscutting concerns as a separate entity, such as an aspect, in which its implementation appears encapsulated only in a part of the program, smooths coupling between modules and increases cohesion of each of them. Moreover, as consequence of low coupling and a high cohesion, the maintainability of the global system improves due to the fact that changes in a module affect only that module; and also the reusability improves due to both base code and aspects can be reused easier in different systems. There are a lot of crosscutting concerns that are usually useful to treat separately and so can be modeled as examples of aspects: logging, authentication, trace, coordination, synchronization, security, persistence, fail-over, error detection and correction, memory management, internationalization, localization, monitoring, product features, data validation, transaction processing, caching, etc.

2.2.2 AO-ADL

AO-ADL is an Aspect-Oriented Architecture Description Language that provides support to separate and inject crosscutting concerns in a non-intrusive way at the architectural level by defining a symmetric composition model, where functional and non-functional concerns are modeled by the same architectural block. AO-ADL also provides support to model parameterized architectural patterns.

A complete description of AO-ADL can be found in [27]. Basically, AO-ADL configurations represent software architectures that contain instances of components and connectors and define how they are attached. Components are the locus of computation and state, and model both base and aspectual behavior. Components are described by means of their ports (or interfaces), which can be provided or required. Composite components are used to encapsulate sub-architectures, defining different levels of abstractions of the components. The main elements of a connector are: roles, component bindings and aspectual bindings. The role element represents provided, required and aspectual interfaces of the connector; the component binding element describes the interaction between provided and required roles; and the aspectual binding element describes the aspectual interactions between provided and aspectual roles. Finally, each interface element is composed by a set of operations with parameters. Ports and roles also have a type that determines the semantics of their operations. Components, composite components, connectors and interfaces can reuse their specification by specifying their implementation, but assigning a different name.

AO-ADL also allows the definition of template models, specifying which elements (components, connectors, interfaces, ...) of an AO-ADL software architecture can be parameterized — i.e., defined as a parameter. AO-ADL distinguishes three different kinds of templates: configuration templates, composite component templates and connector templates. Configuration templates allow the definition of a parameterized software architecture. Composite component templates allow the definition of a parameterized component that can then be instantiated and attached to other components to build a software architecture. It implies the definition of sub-architectures, and in addition sub-templates, with many base and aspectual components for modeling complex quality attributes. Connector templates

encapsulate the weaving information between base and aspectual components.

2.3 Model-Driven Development (MDD)

Model-Driven Development (MDD) [28] is an approach to software development in which the focus and primary artifacts of development are models and model transformations. MDD promotes both the use of models to formally represent domain-specific concepts, and the automation of software development tasks by means of model transformations. These models follow a syntax defined in metamodels. For example, the architecture of a system can be modeled in the UML metamodel by specifying the parts (e.g., components and interfaces) and connectors of the system, and the rules for the interactions of the parts using the connectors. Then, the UML model of the system's architecture can be transformed or translated to any other modeling language or even to programming language (i.e., generating code) by means of model transformations that are specified in a transformation language. Examples of transformation languages are ATL (ATLAS Transformation Language) [29], QVT (Query/View/Transformation)[1], or ETL (Epsilon Transformation Language) [30].

[1] https://www.omg.org/spec/QVT/About-QVT/

Chapter 3

State of the Art and Related Work

This section overviews the current state of art about QAs and in particular FQAs and their management in the context of SPLs. We study the main techniques to manage FQAs, including AOSD and MDD which are used in our approach. We also compare our approach with the related work about FQAs taking into account all phases of the SPL process applied to FQAs. Table 3.1 summarizes the more relevant approaches from the related work to be discussed in this section and compares them with the approach proposed in this thesis.

3.1 Quality Attributes

Most of the approaches that model QAs focus on their analysis as non-functional properties (NFPs) in the generated product of an SPL, such as cost, maintenance, performance, or availability. However, there has been little work devoted to model and manage QAs that have strong functional implications and need to be modeled as functional software components — that is, the modeling of FQAs. This fact is reflected in Table 3.1 (see columns of `QAs Domain Engineering`) where most of the existing approaches deals with the modeling of QAs as NFPs (`Modeling of NFPs`), but only a few of them consider FQAs (`Modeling of FQAs`). Besides, the evolution of the FQAs is ignored in the literature, where only the RiPLE-DE

Table 3.1: Comparison of approaches for modeling QAs in SPLs.

Approach	QAs Domain Engineering						QAs Application Engineering				FQAs Weaving Engineering		
	Modeling of FQAs				Modeling of NFPs								
	FQAs variability	FQAs dependencies	Real implementations	Separation of concern	NFPs variability	Interdependencies between NFPs and FQAs	FQAs customization	Quality aware product configuration	Optimization of configurations	Automation	FQAs weaving	FQAs evolution	Tool support
Definition Hierarchy [32]	✗	✗	✗	✗	✓	✓[2]	✗	✓	✗	✗	✗	~	~
QADA [33]	~	✗	✗	~	~	~	~	✓	✗	✗	✗	✗	✗
Bayesian Belief Network (BBN) [34]	✗	✗	✗	✗	✓	✓[2]	✗	✓	✗	✓	✗	✗	✗
COVAMOF [35]	✓	✓	✗	✗	✗	✓[2]	~	✓	✓	✓	✗	✗	✓
Goal-based model [36]	✗	✗	✗	✗	✓	✓[2]	✗	✓	✓	✓	✗	✗	~
F-SIG [37]	✗	✗	✗	✗	✓	✓[2]	✗	✓	✗	✗	✗	✗	✗
Extended feature model [38]	✗	✗	✗	✗	✓	✓[2]	✗	✓	✓	✓	✗	✗	~
Etxeberria2008 [39]	✓	✗	~	~	✓	✓[1]	✓	✓	~	✗	✗	✗	✗
RiPLE-DE [31]	✓	✗	✗	✗	✓	✓[1]	✓	✓	✗	✗	✗	✓	✗
Analytic Hierarchical Process (AHP) [40]	~	✗	✗	~	✓	✓[2]	✗	✓	✓	✗	✗	✗	✗
CORE [41, 42]	~	✓	✗	✓	✓	✓[2]	✓	✓	✓	✗	✓	✗	✓
Our Approach (WeaFQAs)	✓	✓	✓	✓	✓	✓[1]	✓	✓	✓	✓	✓	✓	✓

✓: Yes. ✗: No. ~: Partially.

1: Real products based. 2: Domain experts' judgments based.

approach [31] mentions it, but it does not provide any support as we do with our evolution algorithms.

3.1.1 Variability modeling of QAs

The main focus of the existing approaches that model QAs is the analysis of how the variations in the functional components of the application affect the NFPs. In [43] Etxeberria et al. gather together and overview different techniques for modeling variability in QAs.

Approaches presented in [32, 34, 36, 37, 38, 44] are based on feature models and extend them in different ways. For instance, In [38], Benavides et al. extend the feature model to deal with extra functional features using attributes, characteristics of a feature that can be measured (e.g., availability, latency) and the relationships between attributes. In [36, 44], González-Baixauli et al. deal with the variant analysis of NFP by introducing a goal/softgoal paradigm and relating it with feature modeling, they also use case modeling. In [37], Jarzabek et al. propose an integrated modeling framework (F-SIG, Feature-Softgoal Interdependency Graph) that extends the feature modeling with concepts of goal-oriented analysis in two ways: (1) to allow developers to record design rationale in the form of inter-dependencies between variant features and QAs during the design of a product line architecture, and (2) to help developers evaluate the impact of variant features selected for a target system during system construction. Zhang et al. [34] use feature models to capture functional requirements of an application while using bayesian belief network (BBN) models to capture the impact of functional variants on the QAs. All of these approaches make analysis of QAs focusing on the achievement degree of NFP (e.g., cost, maintenance) of the final product of an SPL. *But none of them address the variation modeling of the functional part of the QAs themselves.*

Following the approach of annotating the base model by means of extensions to the base modeling language, Rasha Tawhid and Dorina Petriu [45, 46] achieve the variability of the QAs by proposing a technique to model the commonality and variability in structural and behavioral SPL views using Model-Driven Development (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 (e.g., UML profiles with stereotypes such as in [18]) of the QA is derived, and a model for the given product is generated. *Annotating the base model makes it strongly dependent on variability specifications and prevents both the application base model and the QAs model from being reused.* In contrast, using a separate variability language such as CVL allows the independence of the variability language and the modeling language to be maintained. In addition, these approaches model non-functional QAs such as performance instead of FQAs, and introduce the variability at the design level (e.g., within sequences diagrams) instead of modeling the variability of the QAs earlier on in the development process, at the requirement level or at the architectural level.

3.1.2 Non-functional QAs: performance and energy efficiency

Despite the fact that the performance QA has been widely studied in the literature as one of the main QAs to take into account when developing an application [47, 48, 49, 50], energy efficiency (or sustainability) has recently become an important QA to also be considered [51, 52]. Jagroep et al. [51, 52] propose energy efficiency as a new QA, focusing on usage resources such as software utilization, energy usage and workload. Thus, there are several examples of work which analyzes the energy efficiency of different aspects of the applications [53, 54, 55]. However, *little work has been done about the relationships between the energy efficiency and other QAs, and even less on the relationships between the functionalities (i.e., FQAs) required to satisfy traditional QAs (e.g., security, usability) and the energy efficiency of that functionalities and their different configurations [56].*

The performance and energy information of different recurrent functionalities (e.g., the FQAs) is important for analysis in our approach. Thus, we consider several papers that provide the performance and energy experimental information as a repository [57, 58, 59, 60] in order to use the information gathered in our approach and generate the configuration based on that information. The main problem is that none of these repositories provide experimental results on energy

efficiency of FQAs, but rather on other functionalities, like, for example, for the Internet of Things (IoT) components [57], or Java collection classes [61].

The relevance of reasoning about energy efficiency and performance at the architectural level is to be able to compare the energy consumption and performance of the different architectural configurations of the same applications (architectural patterns, design variations, frameworks, etc.) [62, 48, 49, 50]. Some approaches focus on the definition of architectural tactics [51, 52] and design patterns [63] driven by the energy. Other approaches define new architectural description languages (ADLs) that include profiles and analysis of energy consumption and performance [62, 64]. Whatever the case, the experimentation consists of estimating the energy consumption and/or performance of the application code to analyze the effects of applying a specific architectural pattern or design to the application [63, 51, 52, 53]. But, *there is not any approach that considers the different configurations of the recurrent functionality, their parameters and implementations, which can be reused in many different applications, like the FQAs we consider.*

3.2 Functional Quality Attributes

Some QAs have functional implications in the software architecture of the applications affected by them, and can be modeled using software components [6]. In this thesis, this kind of QAs has been coined as Functional Quality Attributes (FQAs), and they refer to the specific functionality that needs to be introduced into the applications to satisfy the desired QAs. Although the concept of FQA has been identified before [65, 5], *none work has paid attention to the complexity of managing FQAs, including their variability and dependencies modeling, their clear separation from the base application and their subsequent incorporation into the base application architecture after customizing them to the requirements of the application.* As depicted in Table 3.1, none work, except our approach presented in this thesis, fully covers all the SPL processes when dealing with QAs (i.e., `QAs Domain Engineering`, `QAs Application Engineering`, and `FQAs Weaving Engineering`).

3.2.1 Identification and characterization of FQAs

Some papers identify FQAs and the necessity of managing their variability [4, 6]. For instance, in [4], the authors analyze around 500 non-functional requirements from different specifications of industrial applications and identify that most of the so called non-functional requirements are not really non-functional because they describe functional behavior of the application. In [6], the authors apply an inductive research method to identify (elicit) FQAs. They identify functionalities related to the usability FQA that affect the software architecture, and define patterns to implement those functionalities in different applications. However, the authors only focus on usability functionalities (e.g., logging, contextual help, feedback,...), while we consider any kind of FQA (e.g., security, persistence,...), including also usability.

3.2.2 Relationships between NFPs and FQAs

Approaches that model QAs can be classified into two categories based on the method used for measuring the interdependencies between FQAs and NFPs [40]:

- **Real products based.** The impacts of individual features on a QA are measured by evaluating software architectures derived via some generic architecture evaluation methods or the execution of final produced products [66, 39]. This kind of approaches collect the information about NFPs by generating and testing the real products in compile-time or runtime, processing the raw results and storing the influence of each feature and the combination of features on a NFP [67, 68]. Requiring the real products and generating a large number of real products for quality evaluation is the main limitation of this kind of approaches because it is costly and time-consuming to generate real products in practice, and therefore, not feasible.

- **Domain experts' judgments based.** Adapting domain experts' judgments for interdependency measurement avoids to generate the real products [40, 69, 37, 34]. These approaches use qualitative or quantitative methods based on using qualitative values to indicate the relative impacts [70, 71] or evaluating the relative impacts numerically with continuous values [40, 34],

respectively. For instance, in [70] the authors use a multi criteria decision making method to analyze the preferences and interactions of QAs based on a fuzzy measure. They define whether two QAs interplay in a complementary way or in a redundancy way. Also, in [71], the authors consider the relationships between QAs, but they only evaluate whether or not they affect other QAs positively or negatively, but they do not quantify the effect.

All these work analyze the relationships between different QAs and functionalities, but they do not distinguish between FQAs and the base functionality of the applications. Indeed, none of them provide a characterization with the purpose of quantifying the QAs. This is because they include neither the support to represent FQAs, nor the configurable characteristics of the frameworks that implement the FQAs. These are essential to analyze the performance and energy consumption of a given architectural configuration. Moreover, none of the work deal with energy efficiency as a QA.

3.2.3 FQAS and Software Product Lines

Existing work that addresses FQAs variability considers that they are part of an SPL and model them jointly with the variability of the base applications. For instance, the RiPLE-DE process [31] (RiSE Product Line Engineering - Design Engineering) and QADA [33] (Quality-driven Architecture Design and quality Analysis). RiPLE-DE [31] is a domain design process for SPL that can be extended to model the FQAs variability as part of a family of products. The variability of the QAs is represented in feature diagrams and in order to achieve desired quality levels, the QAs are enhanced with information of the base application (e.g., the system's response measure). The variation of the attributes is given by that information which is usually numerical values and the architecture is evaluated in order to achieve the necessary variation of the QAs. Thus, the variability of the FQAs directly depends on the base application, avoiding the reuse of the FQAs. QADA [33] is a specific method for designing SPL architectures by transforming systematic functionality and QAs into software architectures, but this proposal does not take into account the quality requirements explicitly.

Concern-Oriented Reuse (CORE) [41, 42] is a reuse paradigm for general-

purpose software development that combines best practices from Model-Driven Engineering (MDE), Component-Based Software Engineering (CBSE), SPL, feature-oriented and aspect-oriented software development, and goal modeling. In CORE, every kind of software characteristic, from base application functionality to non-functional properties, are encapsulated in reusable units called concerns. Although they do not directly manage the concept of FQAs, they identify and encapsulate as concerns the functionality of the FQAs. The main differences between the CORE approach and our approach are that (1) they model the variability of the interfaces of the concerns (e.g., interfaces of frameworks or components) instead of modeling the variability of the internal functionality of the components as we propose; (2) they also focus on the impacts of the concerns on non-functional QAs (e.g., access time, efficiency, etc.) by specifying goals using goal models; while we focus on the functional part of the quality attributes (e.g., the implementation of a particular encryption algorithm and its variants); (3) their approach depends on the Reusable Aspect Models (RAM) weaver. RAM is an AO multi-view modeling approach [72] for software design modeling that consists of a UML package specifying the structure and the behavior of a software design using class, sequence, and state diagrams. So, the RAM weaver is specific for UML models and makes difficult to apply the approach to others ADLs. Our approach, instead, is independent of the language to model the architectures and in the case of CVL, our approach is suitable for using with any MOF-compliant language. Additionally, CVL provides the advantages of Model-Driven Development by allowing us to define custom model transformations to apply any kind of modification to the architecture.

Zhang et al. [40] proposes an approach of modeling QAs in feature models based on domain experts' judgments using the Analytic Hierarchical Process (AHP) and conducting quality aware product configuration based on the captured quality knowledge. They call "contributors" of a QA to the functionality that affects the QAs, but they do not distinguish between FQAs and the application functionality, and therefore, they model variability of FQAs jointly with the variability of the base application.

Another interesting approach is presented in [67]. The authors approximate the influence of each feature in the feature model on a non-functional property, before generating the configurations. However, they predict the effects of the features

instead of giving real measurements. Additionally, they model the applications' variability, so they need to build a variability model for each different application, while we focus on specific recurrent functionality (FQAs). Their variability model is always the same because the FQAs can be reused in several applications. Furthermore, they do not consider energy efficiency of applications in their work, which is also a novel and well-known non-functional property nowadays.

3.2.4 Dependency modeling of FQAs

The support for modeling dependencies already exist at different levels (e.g., in feature models). For instance, in [73] the authors achieve the dependency modeling in the context of feature modeling. They decompose an overall diagram into a set of individual feature models, and propose a matrix-based approach to maintain and manage the information about feature dependencies between different feature model trees. Encoding the dependencies in an auxiliary structure such as a matrix improves their automatic management through matrix operations, but it complicates the knowledge about them from the point of view of the domain experts who should specify the dependencies explicitly in order to ensure that the dependencies will be noticed by the application architects.

One framework that models variability on all layers of abstraction of an SPL including the dependency relationships is COVAMOF [74, 35]. COVAMOF captures variability of FQAs in terms of variation points and dependencies by using associations. Dependencies specify properties for the feature models that define values of the QAs such as performance or memory usage. Other approaches that take into account dependencies of FQAs are [75] and [76]. Egyed and Grunbacher [75] propose automated traceability techniques to eliminate falsely identified conflicts and cooperation efficiently between the quality attributes. In [76] the authors propose a quantitative quality-driven approach that attempts to find the best possible fit between conflicting stakeholders' quality goals, competing architectural concerns, and project constraints. The problem of these approaches is that model the FQAs' variability jointly with the variability of the application making more difficult the management, customization, and evolution of the FQAs.

3.2.5 FQAs and separation of concerns

Although a large number of Aspect-Oriented (AO) approaches have emerged in recent years at architectural level of the software life cycle, there has been little work done to model the variability of the QAs using AO techniques. A significant proposal is the DiVA project[1], which provides tools and AO model-driven methodologies, covering all stages of the software life-cycle, from requirement analysis to execution. This approach is oriented to modify the software architecture at runtime. McVeigh et al. [77] propose an approach based on component resemblance in which base components can be modified in order to reuse them without affecting the rest of component users. 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.

Other approaches that address variability at the architectural level using SPL are [78] and [79]. In [78], 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. Finally, a hierarchical variability modeling mechanism is proposed in [79]. 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. However, none of these approaches directly use AO techniques, and thus, cannot provide the same advantages obtained when using AOSD as discussed in Section 2.2.

3.2.6 FQAs and Model-Driven Development

Model Driven Development (MDD) has also been used in the field of SPLs and QAs [80]. Sijtema proposes a strategy to let the ATL Transformation Language [29] handle the variability by extending the concrete syntax of ATL with the concept of variability rules. Variability rules are used in the context of a transformation

[1] http://www.ict-diva.eu

sequence which successively refines models. However, they first model the variability separately in a feature diagram and have to make the correspondence between the feature selections and the realization of the artifacts. One of the advantages of CVL is that it also allows to use MDD to extend the semantics of the variation points with other transformation languages such as QVT (Query/View/Transformation) or ETL (Epsilon Transformation Language) [30].

Table 3.1 summarizes the characteristics supported by existing approaches in the development life cycle of the FQAs. We have only included in Table 3.1 those approaches that deal with QAs as part of an SPL [33, 34, 35, 36, 37, 38, 39, 31, 40, 41, 42]. While all of them model QAs as non-functional properties, only a few of them [35, 39, 31, 41, 42] directly deal with the functional parts of the QAs. The CORE approach [41, 42] is the only one that covers almost all engineering processes (that is, the (`QAs Domain Engineering`, `QAs Application Engineering`, and `FQAs Weaving Engineering`), but CORE does not consider the concept of FQA because all features of a system (both functional or non-functional) are modeled as "concerns" in CORE. In addition, this approach does not take into account the evolution process when the requirements change in order to update the FQAs. Finally, there is a lack of tool support since existing tools focus on the variability modeling using feature models and the analysis of those models, but no tool supports the weaving process nor the evolution of FQAs.

Chapter 4

Motivation and Challenges

This section defines the specific challenges addressed in this thesis related to FQAs. The motivation that justifies the consideration of each challenge is briefly described before introducing the challenge.

- **Challenge 1. Managing the complexity of FQAs.**
  Most FQAs are very complex, as they are composed by many concerns that can be related with each other (e.g., security that includes the privacy, confidentiality, and authentication concerns among others). The dependency relationships within an FQA and between different FQAs often go unnoticed by the software architects, who are not domain experts in modeling FQAs. Moreover, there are several frameworks and third party libraries that provide different implementations of FQAs ready to be reused, such as the Java Security package, the Apache Commons library, and the Spring Framework. 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 of the base application [81].

  The challenge is to identify, characterize, and formally model all the possible FQA variation points and their dependency relationships. This should be done from the early stages in the development process, and independently of the final application that requires them, which is not a trivial task.

- **Challenge 2. Analyzing the relationships between FQAs and non-functional QAs.**

Although there is much variability in FQAs, not all variants and implementations fulfill the system's quality requirements in the same way. Each existing framework or library provides different degrees of quality, at the expense of a higher or lower use of resources, and therefore different level of non-functional QAs (e.g., energy efficiency, performance). Unfortunately, there are not enough experimental studies that show how each FQA solution (i.e., what parameters and variables configuration present in the FQA) addresses the non-functional requirements and if choosing a particular solution that satisfy a QA (e.g., energy efficiency) will affect other QA (e.g., performance). In general, developers are not aware of the implications of architectural decisions on a non-functional QA [51], and they need some help to make the correct design decisions from the point of view of such QA, without penalizing the other QAs.

The challenge is to analyze which and how variables and parameters of the FQAs affect non-functional requirements, in particular performance and energy efficiency. This includes gathering the information through experimentation and modeling it as part of the FQAs' models.

- **Challenge 3. Defining an automatic process to generate customized FQAs for each application.**
  FQAs are recurrent and have many points of variability. However, since there are many variation points, not all of the concerns of an 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.

  The challenge is to define an automate process to model the commonalities and variabilities of the FQAs, and to generate automatically different configurations of the FQAs to the requirements of each different application.

- **Challenge 4. Generating optimum configurations of FQAs based on non-functional QAs.**
  When customizing the FQAs it is recommendable to also consider the influence of the new components on other non-functional QAs, like performance or energy efficiency. However, finding the optimum configuration of the FQAs that satisfies all the application's requirements and that additionally takes into account non-functional QAs complicates the process.

  The challenge is to define a process that helps application architects and developers to automatically generate optimum configurations of FQAs in terms of non-functional QAs (e.g., performance and energy efficiency).

- **Challenge 5. Achieving separation of concerns between FQAs and the 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 logging FQA to provide feedback crosscuts all the points of the application in which some kind of feedback information needs to be provided to the user.

  The challenge here is to separate the modeling of the FQAs from the base application. This will allow the later incorporation of customized FQAs into the applications in a non-intrusive way — i.e., without manually modifying the base application.

- **Challenge 6. Weaving the customized FQAs with the final application.**
  After generating a customized configuration of the FQAs, the resulting models need to be combined (woven) with the models of the final application, which are specified using a particular modeling language. This is not a straightforward task because each FQA needs to be woven at different points of the base application, and multiple views may be required to appropriately model the FQAs (e.g., behavioral view, structural view). Moreover, appli-

cations 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. This makes the weaving process even more complex.

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., architectural components should be completely unaware of the FQAs they are affected by. Therefore, this challenge includes the definition of architectural patterns with reusable FQAs.

- **Challenge 7. Providing support for the evolution of FQAs.**
  FQAs can evolve with the applications in the future (e.g., security can change over time to include most sophisticated encryption or authentication algorithms). Evolving the FQAs also implies: (1) to elicit the changes that need to be performed within the deployed configurations of FQAs when the application's requirements or the FQAs' technology evolve; and (2) to obtain a new valid, evolved software architecture of the FQAs consistent with the configuration already deployed for the FQAs.

  The challenge is to define an automatic process for adapting and evolving the FQAs and their deployed configurations. Also, the automatic evolution process will be useful only if it is correct and efficient for a large number of configurations so it is important to demonstrate the efficiency and correctness of the evolution process.

- **Challenge 8. Supporting the approach with tools.**
  The approach presented in this thesis will not be viable without the required tool support.

  The challenge is to develop tools to automate the process of (1) generating customized software architectures for the FQAs required by an application, (2) weaving these software architectures with the architecture of the base ap-

plication; and (3) managing the evolution of the FQAs when the requirements change or the FQAs' technology is updated.

- **Challenge 9. Evaluating the approach.**
  The approach needs to be evaluated both quantitatively by using appropriate metrics to assess the benefits of our approach, and qualitatively, when the use of a metric does not make sense. Also, the approach needs to be applied to real case studies to evaluate its applicability.

  The challenge is to evaluate quantitatively and qualitatively the proposal using real case studies.

Chapter 5

Approach Overview

In this chapter we present a general overview of the WeaFQAs approach. We also present the two different implementations of WeaFQAs developed in the thesis, as well as the main tool that supports the approach.

5.1 WeaFQAs

Our research focuses on modeling and customizing FQAs separately from the applications, and then incorporating the FQA configurations into the applications by combining the benefits provided by several technologies: Software Product Lines (SPLs), Aspect-Oriented Software Development (AOSD) and Model-Driven Development (MDD). We propose an Aspect-Oriented Software Product Line (AO-SPL) approach, called *WeaFQAs*, that extends the classic framework for SPL engineering [7] as shown in Figure 5.1. WeaFQAs distinguishes between `Domain`, `Application`, `Weaving`, and `Evolution Engineering` processes that we describe in the following sections. Note that these engineering processes of WeaFQAs are focused on modeling and managing FQAs, but they do not focus on modeling the base functionality of the applications. Despite this, WeaFQAs needs to consider the application requirements about QAs and its software architecture in order to incorporate the FQAs.

Within WeaFQAs two different roles are defined: (1) *domain experts*, who are in charge of defining the assets of the `Domain Engineering` process, and

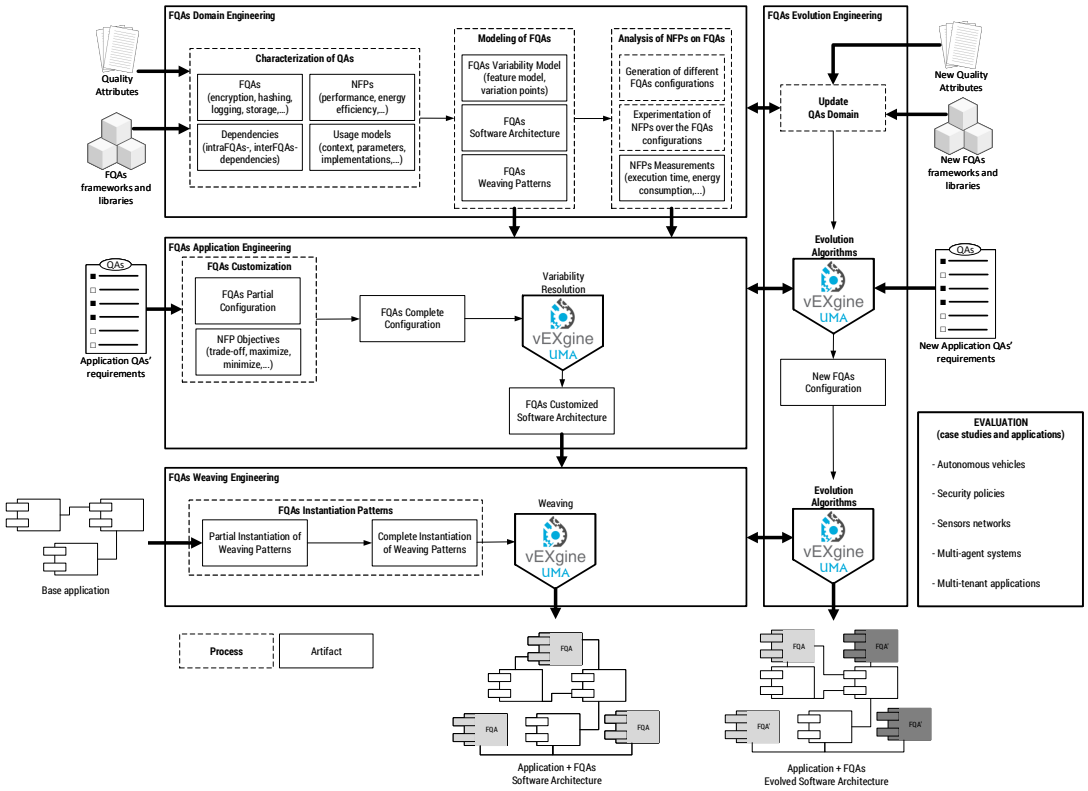

Figure 5.1: Overview of the WeaFQAs approach.

(2) the *application engineers* or *software architects*, who use those assets, reusing and instantiating them for each application in the `Application` and `Weaving Engineering` process. Note that domain experts do their work only once. Application engineers reuse the work done by the domain experts when they use the SPL to generate specific configurations according to application requirements. In the `Evolution Engineering` process, both domain experts and application engineers take parts. While domain engineers are in charge of updating the reusable artifacts of the `Domain Engineering` process with new changes, application engineers are in charge of providing the new application requirements in order to automatically propagate the evolution changes through the artifacts of the `Application` and `Weaving Engineering` processes.

5.1.1 FQAs Domain Engineering

The `FQAs Domain Engineering` process (top of Figure 5.1) manages the complexity and the variability of the FQAs from the early stages of the development life cycle. The goals of our domain engineering process are: (i) to characterize the QAs and the FQAs (`Characterization of QAs` in Figure 5.1); (ii) to define the commonality and the variability of the FQAs including their dependencies, and construct reusable FQA solutions (e.g., the software architecture of the FQAs and their weaving patterns) that accomplishes the desired variability (`Modeling of FQAs`); and (iii) to analyze the influence of the different FQA variants in the non-functional QAs or non-functional properties (NFPs) — i.e., to analyze the interdependencies between FQAs and NFPs (`Analysis of NFPs on FQAs`).

This process takes as input the available information about the QAs domain such as research papers, surveys, or QAs' catalogs (`Quality Attributes` input in Figure 5.1) and the existing implementation of FQAs such as real frameworks and third-party libraries (`FQAs framework and libraries`). From this input, domain experts identify the commonality and variability of the FQAs as well as the existing dependency relationships between FQAs, the usage models of the FQAs, and non-functional properties (`Characterization of QAs`). The usage model of an FQA is defined by means of a set of variables that can have a positive or negative effect on other QAs that are relevant for the application under development (e.g., energy efficiency and performance), and the values that each variable can take. For instance, the characterization of the encryption FQA includes (i) identifying the different encryption algorithms available by the existing security frameworks, along their variables and parameters such as the key length, block cipher mode of operation, and padding; (ii) the dependency relationships between encryption and others FQAs (e.g., hashing); and (iii) the usage model that for the encryption FQA includes the size of the object or file to be encrypt or decrypt, as well as those variable and parameters specific of the encryption functionality (e.g., the block cipher mode) that may affect the NFPs (e.g., performance).

The FQAs characterization allows a variability model to be specified and defined for the FQAs and a software architecture to be constructed that supports

the variability (`Modeling of FQAs`). Reusable architectural patterns for weaving the FQAs are also specified by the domain experts in order to define how the different FQAs should be composed with the core architecture of the base application. In addition, the FQAs' models are enriched with NFPs information such as energy efficiency and performance information gathered through empirical analysis and/or experimentation (`Analysis of NFPs on FQAs`). This information will be then used during the application engineering process to generate optimum configurations of the FQAs.

An important thing that is worth highlighting in the WeaFQAs approach is that artifacts of the domain engineering process will be completely reused by any application that wants to configure and incorporate these FQAs into its software architecture. Those artifacts are the `FQAs Variability Model`, the `FQAs Software Architecture`, the `FQAs Weaving Patterns`, and the NFPs information gathered by experimentation (`NFPs Measurements`).

5.1.2 FQAs Application Engineering

The `FQAs Application Engineering` process (middle of Figure 5.1) aims to generate FQA configurations based on the specific application's requirements. The goal of the application engineering process is to bind the FQAs variability according to the application requirements (`FQAs Customization` in Figure 5.1).

To accomplish this goal, the application engineer identifies the QAs required by the application (`Application QAs' requirements`) and creates a product configuration of the FQAs that fulfills those requirements (`FQAs Customization`). This is done by selecting the features that satisfy the application's QAs requirements — i.e., by instantiating the FQAs variability model, including the FQAs' usage models, previously specified during the domain engineering process. The application engineer also selects an objective function (defined during the domain engineering process) to generate the optimum FQA configuration (e.g., one that maximizes the energy efficiency). Often, the application engineer can only provide a partial instantiation of the variability model and the usage models, so a partial configuration is generated in these cases (`FQAs Partial Configuration`). Both this partial configuration and the objective function (`NFP Objectives`)

are used within the NFPs' information gathered in the domain engineering process to automatically generate a complete optimum FQA configuration (`FQAs Complete Configuration`). This alleviates the application engineer's task of deciding between different configurations.

Then, an FQAs architecture configuration is automatically generated as the realization of the product configuration — i.e., the variability resolution. The variability resolution is automatically performed with the `vEXgine` tool [82] that is a customizable and extensible execution engine to resolve the variability, and built to support the WeaFQAs approach. The output of the application engineering process is an architecture configuration of the FQAs (`FQAs Customized Software Architecture`) that only contains those artifacts of the FQAs software architecture that are needed according to the application requirements.

5.1.3 FQAs Weaving Engineering

The `FQAs Weaving Engineering` process (bottom of Figure 5.1) is in charge of integrating the FQAs architecture configuration generated in the previous process into the core architecture of the application being built. The goal of the weaving engineering process is to incorporate the customized architectural model of the FQAs into the base application by applying the appropriate weaving patterns (`FQAs Instantiation Patterns` in Figure 5.1). In WeaFQAs, the weaving engineering process is based on AOSD technologies. AOSD enables weaving 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.

The weaving process is not a straightforward task since each FQA has to be woven at different points of the base applications (join points). Furthermore, each FQA will be woven according to a different weaving pattern, depending on the semantics of the FQA. Thus, the application engineer must identify the points in the application where the FQAs will be incorporated (`Base Application`), and associate the set of FQAs weaving patterns provided as part of the FQAs domain engineering process with the customized components of the FQAs architecture (`FQAs Instantiation Patterns`). The application engineer can provide a partial in-

stantiation of the weaving patterns (`Partial Instantiation of Weaving Patterns`) because the lack of information in the application's requirements about the places where to introduce the FQAs. In those cases, WeaFQAs will automatically identify all points in the application architecture where would be possible to apply the specific FQA weaving pattern. With that information, application engineer is able to provide a complete instantiation of the FQA weaving pattern (`Complete Instantiation of Weaving Patterns`).

Also, the weaving process is automatically performed with the `vEXgine` support, without manually modifying the core architecture of the application. The output of this weaving process is a software architecture of the application that also incorporates the required FQAs (`Application + FQAs Software Architecture`).

5.1.4 FQAs Evolution Engineering

The `FQAs Evolution Engineering` process (right side of Figure 5.1) addresses the managing of the FQAs when the requirements of the application change and/or the technology of the FQAs evolves. The goal of this process is to update the artifacts of the different engineering process in the WeaFQAs approach (`Update QAs Domain` in Figure 5.1) and propagate the changes to the final application already deployed (`Evolution Algorithms`).

The inputs of this process are: (i) the new QAs that can appear in the future as for example the most recent energy efficiency QA (`New Quality Attributes`); (ii) new implementations of the FQAs or updates of the existing frameworks and libraries (`New FQAs frameworks and libraries`), as for example a new version of the Spring framework; and (iii) the new QAs' requirements of the application that need to be taking into account (`New Application QAs' requirements`). The first two kind of inputs require the domain experts to manually update the artifacts of the domain engineering process in order to incorporate the possible new products to the SPL to make them available to the applications. The third input requires the application engineer to provides a new configuration with the new requirements.

In all cases, the changes need to be automatically propagated to the previous

deployed configurations of FQAs. To automatize this task, WeaFQAs provides a set of evolution algorithms to calculate difference between configurations and update the previous configuration (`New FQAs Configuration`), but also an evolution algorithm to propagate the changes to the software architecture of the FQAs, obtaining as result the final application with the FQAs evolved (`Application + FQAs Evolved Software Architecture`).

5.2 Implementations of WeaFQAs

Since variability can be expressed through multiple techniques such as feature models [9], annotations [18, 19] or by using a variability language such as CVL [21], in this thesis we provide and compare two different implementations of our generic approach using different variability and architecture description languages:

- Using traditional feature models to specify the variability of the FQAs [9], and AO-ADL, an aspect-oriented architecture description language [27], to model the software architecture of the FQAs. This implementation uses also a variability modeling language (VML) [83] to link the features of the feature model with the architectural elements of the FQA models.

- Using CVL [21] to specify the variability and variation points of the FQAs, and a MOF-compliant language such as UML to model the software architecture of the FQAs.

These two instantiations of the WeaFQAs approach are presented and compared in detail in Chapter 8.

5.3 vEXgine

`vEXgine`[1] [82] is the tool developed within the scope of this thesis to support the WeaFQAs approach. vEXgine is a customizable and extensible implementation of the execution engine for the Common Variability Language (CVL). It has been

[1] `http://caosd.lcc.uma.es/vexgine`

developed to solve lack of tools that support the CVL approach. The tool fully supports the materialization process of CVL, including the delegation mechanism that can be extended with different delegation engines. It includes an implementation of the delegation engine based on Model-to-Model transformations using a general purpose transformation language such as ATL. The provided Java API of the execution engine allows extending the CVL approach to fulfill the industrial needs for variability modeling in SPLs.

Chapter 6

Discussion of Results

This chapter discusses the contributions of this thesis and summarizes the publications composing the PhD thesis describing how they address the aforementioned challenges.

6.1 Contributions

Within the scope of this thesis, the main contribution is WeaFQAs, an SPL approach to model, customize, and incorporate FQAs independently from the applications. WeaFQAs addresses most of the challenges (Challenges 1-8) defined in Chapter 4, while Challenge 9 has been addressed evaluating WeaFQAs qualitatively and quantitatively, but also applying it to several case studies and real applications.

This section specifies the main publications where the contributions of WeaFQAs are split:

1. A characterization of QAs specially focusing on FQAs. This includes the identification of the variability and dependency relationships of the FQAs, and the modeling of a customizable software architecture of FQAs and the architectural patterns to integrate them into the applications. This contribution is presented in [84] (Chapter 8).

2. An automatic process to model FQAs and generate customized configurations of them based on the requirements of each application. This con-

tribution corresponds with the first version of WeaFQAs presented in [84] (Chapter 8).

3. Analysis of the variables and parameters of the FQAs that affect the performance and energy efficiency of the applications. This also includes the definition of usage models based on real implementation of FQAs, the experimentation, and the energy and performance information gathered that can be used in other researches. This contribution is presented in [85] (Chapter 9).

4. A process to help application engineers and developers to automatically generate optimum configurations of FQAs in terms of performance and energy efficiency. This contributions is presented in [85] (Chapter 9).

5. Separate the modeling of FQAs from the applications achieving separation of concerns between FQAs and the applications. This contribution can be found in the whole approach of WeaFQAs using AOSD, specially in [84] (Chapter 8), in [85] (Chapter 9), and in [86] (Chapter 10).

6. A process that systematically weave FQA configurations into the base architecture of a given application, without the needs of manually modifying the application. This contribution is presented in [84] (Chapter 8) as part of the WeaFQAs approach and in [86] (Chapter 10) as an implementation of this process and as an application of WeaFQAs in the context of security policies.

7. An automatic process for evolving FQAs and their deployed configurations when the applications requirements changes and/or the FQA technology evolves. This contribution is entirely presented in [87] (Chapter 11).

8. A tool to automate the process of generating customized software architectures of FQAs and weaving these software architectures with the applications. Concretely, we have developed the *vEXgine* tool, that is presented in [82]. Although this publication is not part of the selected publication composing the PhD thesis, it is directly related with our work and it was award-winning the best paper of the tools track in its conference.

9. Applying the approach to multiple real case studies and applications in the industry. In particular, WeaFQAs has been applied in the context of autonomous vehicles (this contribution is presented in [88] - Chapter 12). But, the approach has also been applied in the context of security policies [89], sensors networks [90], multi-tenant applications in the cloud [87], and multi-agent systems [91]. These applications are part of the evaluation of WeaFQAs showing the applicability of our approach. The latter was award-winning the best paper of the conference.

6.2 Compendium of publications of the thesis

A total of 23 research articles has been published as the research work of this thesis (see Appendix A), of which we have selected the following five publications that comprises the main contributions of the PhD thesis and that are presented in the following chapters:

- **Chapter 8.** *An automatic process for weaving functional quality attributes using a software product line approach* [84]. This publication presents the WeaFQAs approach covering the modeling of FQAs, including their dependency modeling, the customization of the FQAs to the application requirements, and the weaving of the FQAs and the applications, including the definition of architectural (weaving) patterns of FQAs. This publication also presents and compares two implementations of the WeaFQAs approach, and evaluates qualitatively and quantitatively the approach.

- **Chapter 9.** *Variability models for generating efficient configurations of functional quality attributes* [85]. This publication extends the WeaFQAs approach to generate optimum configurations of FQAs based on non-functional properties, in particular, energy efficiency and performance. This includes the definition of usage models of real implementations of FQAs, the experimentation over those implementation to obtain the energy and performance information, and the subsequent use of that information to optimize the configurations.

- **Chapter 10.** *Runtime enforcement of dynamic security policies* [86]. This publication presents the applicability of the WeaFQAs approach in the context of dynamic security policies where the security FQAs is main focus. It includes an implementation of the different processes using AOSD and AOP and integrates them inside the INTER-TRUST framework [92].

- **Chapter 11.** *Product line architecture for automatic evolution of multi-tenant applications* [87]. This publications covers the evolution engineering process of the FQAs by applying it in the context of multi-tenant applications, and where the recurrent functionalities play the role of FQAs.

- **Chapter 12.** *Context-dependent reconfiguration of autonomous vehicles in mixed traffic* [88]. This publication is another applicability of the WeaFQAs approach in the context of the autonomous vehicles. In this case, WeaFQAs is adapted to model the behavior of autonomous vehicles and improve the QAs related to traffic (efficiency and safety). This publications is a collaboration with the Distributed Systems Group of the School of Computer Science and Statistics of the Trinity College of Dublin (TCD) in Ireland, as part of the internship to apply for the mention of "International PhD" of this thesis.

Apart form these main publications, Appendix A lists all publications done in the context of this thesis.

6.3 Research Projects

The research objectives of this thesis has been also part of an European project and several Spanish projects. In particular, the European Interoperable Trust Assurance Infrastructure (INTER-TRUST) FP7-317731 project aims to develop a dynamic and scalable framework to support trustworthy services and applications in heterogeneous networks and devices, based on the enforcement of interoperable and changing security policies at runtime by using Aspect-Oriented Programming (AOP), addressing the needs of developers, integrators and operators. So, with this project our research has been benefited from experience of security experts, so security is one of the FQA we were more focused on. The national Spanish

TIN2012-34840 project "Modelos, Aspects, y Variabilidad aplicados a la auto-adaptación en la Internet de las cosas" (MAVI) covers the variability modeling of our approach that we apply to FQAs. Aspect-Orientation and evolution of the FQAs is also part of this project objectives. The regional project "MAGIC: Líneas de producto software y sistemas Multiagente para la Auto-Gestión de sistemas de la Internet-de-las-Cosas" P12-TIC1814 has similar goals to the national project but in the multi-agent context. Since the goal of the project is to provide a generic solution, our process should be able to inject customized FQAs to software agents metamodels. Finally, the project "HADAS: Herramienta de Análisis y Desarrollo de Aplicaciones Sostenibles" defines a sustainable development process capable of generating ecoefficient applications that later can be adapted dynamically to the real energy consumption, and thus, part of the goal of this project is directly related with the process of generating optimum configurations of FQAs in WeaFQAs.

Chapter 7

Conclusions and Future Work

This chapter presents the conclusions of the thesis and the future work.

7.1 Conclusions

Our research has focused on defining an automatic process following an aspect-oriented software product line (AO-SPL) approach that pursues: (i) to separate the modeling of the FQAs from the base applications; (ii) to generate customized FQAs adapted to applications requirements and optimizing non-functional properties; (iii) to incorporate customized FQAs into the applications in a non-intrusive way by defining aspect-oriented weaving patterns; and (iv) to give support for the evolution of application driven by FQAs. As result, the WeaFQAs approach has been defined, instantiated, evaluated, and applied to multiple case studies. The WeaFQAs approach combines SPLs, AOSD and MDD software technologies.

By separating the modeling of the FQAs from the base applications our proposal improves both the modularity and reusability of FQAs. AOSD aims to achieve this separation of FQAs by considering them as crosscutting concerns. Evolution and injection of code can be also improved by different kinds of aspect-oriented weaving provided by the existing Aspect-Oriented Programming (AOP) frameworks. The separation of concerns also facilitates the modeling, design and implementation of both FQAs and base applications. On the other hand, SPLs helps to manage the FQAs from the early stages in the development process, by

facilitating the modeling of the FQAs variability and the process of generating the different FQAs configurations customized for the applications. The use of SPLs improves the maintainability of the software architectures, but we have focused on CVL since it increases the scalability and reusability of variability models in comparison to feature models.

We expect our approach allows: (1) improving the modularization and reusability of the FQAs thanks to AOSD increasing the quality of applications; and (2) improving the maintainability and extensibility of the FQAs thanks to the SPLs.

7.2 Future Work

For future work, we plan to continue researching on top of the WeaFQAs approach to provide full support for the runtime adaptation and dynamic reconfiguration of FQAs and the applications that require them. 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. So, we plan endowing FQAs and applications with dynamicity and self-management capacities using the technology and benefits of Dynamic Software Product Lines (DSPLs) [93] as for example, making the FQAs' models available at runtime to allow their dynamic customization (e.g., using models@runtime).

In addition, we also plan to extends WeaFQAs to consider other non-functional properties of the different configurations of FQAs, together with the energy efficiency and performance attributes, such as memory consumption, or qualitative attributes like levels of security and levels of usability. Along this, a problematic that arises is optimizing several quality attributes simultaneously, specially when multiple FQAs and NFPs are interactions between them. Therefore, we also plan to study some multiobjective techniques for the generation of the FQAs' configurations. For instance, goal-oriented programming, compromise programming, or evolutionary algorithms.

For some FQAs, generating all the configurations and performing all the experiments within a reasonable time may be an intractable task, even more when the configurations needs to be generated at runtime. In those cases, scalability could be an issue of the WeaFQAs approach, so we need to find optimum ways to generate and manage the FQAs' configurations, and analyze NFPs of those configurations when using real implementations of the FQAs.

Finally, we expect to apply the acquired knowledge of the technologies used (e.g., AOSD, SPLs, MDD, CVL, feature models,...) to others software engineering fields, not only on the QAs domain.

Part II

Thesis Publications

Chapter 8

An Automatic Process for Weaving FQAs using an SPL Approach

Title:	An automatic process for weaving functional quality attributes using a software product line approach
Authors:	José Miguel Horcas, Mónica Pinto, Lidia Fuentes
Journal:	Journal of Systems and Software (JSS)
JCR Impact Factor:	2.444 (Q1)
Publication Date:	February 2016
DOI:	https://doi.org/10.1016/j.jss.2015.11.005

Abstract. Some quality attributes can be modeled 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 Prod-

uct 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. Modeling FQAs separately from the base application has many advantages (e.g., reusability, less coupled components, high cohesive architectures).

Chapter 9

Variability Models for Generating Efficient Configurations of FQAs

Title:	Variability models for generating efficient configurations of functional quality attributes
Authors:	José Miguel Horcas, Mónica Pinto, Lidia Fuentes
Journal:	Information and Software Technology (IST)
JCR Impact Factor:	2.694 (Q1)
Publication Date:	March 2018
DOI:	https://doi.org/10.1016/j.infsof.2017.10.018

Abstract. Quality attributes play a critical role in the architecture elicitation phase. Software Sustainability and energy efficiency is becoming a critical quality attribute that can be used as a selection criteria to choose from among different design or implementation alternatives. Energy efficiency usually competes with other non-functional requirements, like for instance, performance. This paper presents a process that helps developers to automatically generate optimum configurations of functional quality attributes in terms of energy efficiency and performance. Functional quality attributes refer to the behavioral properties that need to be incorporated inside a software architecture to fulfill a particular quality attribute (e.g., encryption and authentication for the security quality attribute,

logging for the usability quality attribute). Quality attributes are characterized to identify their design and implementation variants and how the different configurations influence both energy efficiency and performance. A usage model for each characterized quality attribute is defined. The variability of quality attributes, as well as the energy efficiency and performance experiment results, are represented as a constraint satisfaction problem with the goal of formally reasoning about it. Then, a configuration of the selected functional quality attributes is automatically generated, which is optimum with respect to a selected objective function. Software developers can improve the energy efficiency and/or performance of their applications by using our approach to perform a richer analysis of the energy consumption and performance of different alternatives for functional quality attributes. We show quantitative values of the benefits of using our approach and discuss the threats to validity. The process presented in this paper will help software developers to build more energy efficient software, whilst also being aware of how their decisions affect other quality attributes, such as performance.

Chapter 10

Runtime Enforcement of Dynamic Security Policies

Title:	Runtime enforcement of dynamic security policies
Authors:	José Miguel Horcas, Mónica Pinto, Lidia Fuentes
Conference:	8th European Conference on Software Architecture (ECSA)
Conference Rating:	GGS: B / CORE: A / LiveSHINE: B / MA: C
Publication Date:	August 2014
DOI:	https://doi.org/10.1007/978-3-319-09970-5_29

Abstract. The security policies of an application can change at runtime due to several reasons, as for example the changes on the user preferences, the lack of enough resources in mobile environments or the negotiation of security levels between the interacting parties. As these security policies change, the application code that copes with the security functionalities should be adapted in order to enforce at runtime the changing security policies. In this paper we present the design, implementation and evaluation of a runtime security adaptation service. This service is based on the combination of autonomic computing and aspect-oriented programming, where the security functionalities are implemented as aspects that are dynamically configured, deployed or un-deployed by generating and executing

a security adaptation plan. This service is part of the INTER-TRUST framework, a complete solution for the definition, negotiation and run-time enforcement of security policies.

Chapter 11

Product Line Architecture for Automatic Evolution of Multi-Tenant Applications

Title:	Product line architecture for automatic evolution of multi-tenant applications
Authors:	José Miguel Horcas, Mónica Pinto, Lidia Fuentes
Conference:	IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)
Conference Rating:	GGS: B / CORE: B / LiveSHINE: A- / MA: B
Publication Date:	September 2016
DOI:	https://doi.org/10.1109/EDOC.2016.7579384

Abstract. Cloud computing is becoming the predominant mechanism to seamlessly deploy applications with special requirements such as massive storage sharing or load balancing, usually provided as services by cloud platforms. A developer can improve the application's delivery and productivity by following a multi tenancy approach, where variants of the same application can be quickly customized to the necessities of each tenant. However, managing the inherent variability existing in multi-tenant applications and, even more importantly, managing the evolution

of a multi-tenant application with hundreds of tenants and thousands of different valid architectural configurations can become intractable if performed manually. In this paper we propose a product line architecture approach in which: (1) we use cardinality-based variability models to model each tenant as a clonable feature, (2) we automate the process of evolving the multi-tenant application architecture, and (3) we demonstrate that the implemented process is correct and efficient for a high number of tenants in a reasonable time. We use a running case study in the domain of medical software.

Chapter 12

Context-Dependent Reconfiguration of Autonomous Vehicles in Mixed Traffic

Title:	Context-dependent reconfiguration of autonomous vehicles in mixed traffic
Authors:	José Miguel Horcas, Julien Monteil, Mélanie Bouroche, Mónica Pinto, Lidia Fuentes, Siobhán Clarke
Journal:	Journal of Software: Evolution and Process (JSEP)
JCR Impact Factor:	1.033 (Q3)
Publication Date:	April 2018
DOI:	http://onlinelibrary.wiley.com/doi/10.1002/smr.1926/full

Abstract. Human drivers naturally adapt their behavior depending on the traffic conditions, such as the current weather and road type. Autonomous vehicles need to do the same, in a way that is both safe and efficient in traffic composed of both conventional and autonomous vehicles. In this paper, we demonstrate the applicability of a reconfigurable vehicle controller agent for autonomous vehicles that adapts the parameters of a used car-following model at runtime, so as to maintain a high degree of traffic quality (efficiency and safety) under different

weather conditions. We follow a Dynamic Software Product Line (DSPL) approach to model the variability of the car-following model parameters, context changes and traffic quality, and generate specific configurations for each particular context. Under realistic conditions, autonomous vehicles have only a very local knowledge of other vehicles' variables. We investigate a distributed model predictive controller agent for autonomous vehicles to estimate their behavioral parameters at runtime, based on their available knowledge of the system. We show that autonomous vehicles with the proposed reconfigurable controller agent lead to behavior similar to that achieved by human drivers, depending on the context.

Part III

Appendices

Appendix A

Publications

This chapter presents all the publications done in the context of the thesis. The content of these publications is part of the contributions of this thesis, or is directly related with this thesis. As shown in Figure A.1 and A.2 there are 23 publications in total along the 5 years of PhD (2013-2018) divided as follows: 5 journals JCR, 12 international conferences, 2 workshops in international conferences, 2 national conferences, 1 tutorial, and 1 doctoral symposium. Table A.1 lists the publications in journals, Table A.2 and A.3 list the publications in international conferences, Table A.4 lists the publications in workshops and others, and Table A.5 lists the publication in national conferences. It is worthy to highlight the award-winning best papers in two international conferences, one of this corresponds with the tool that provides support to the WeaFQAs approach.

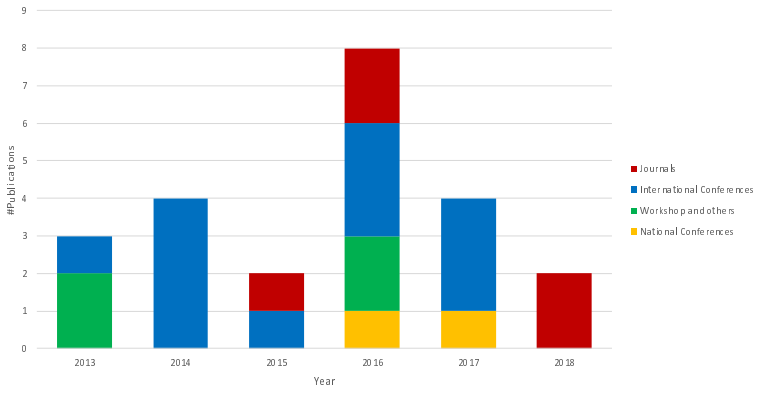

Figure A.1: Publication productivity.

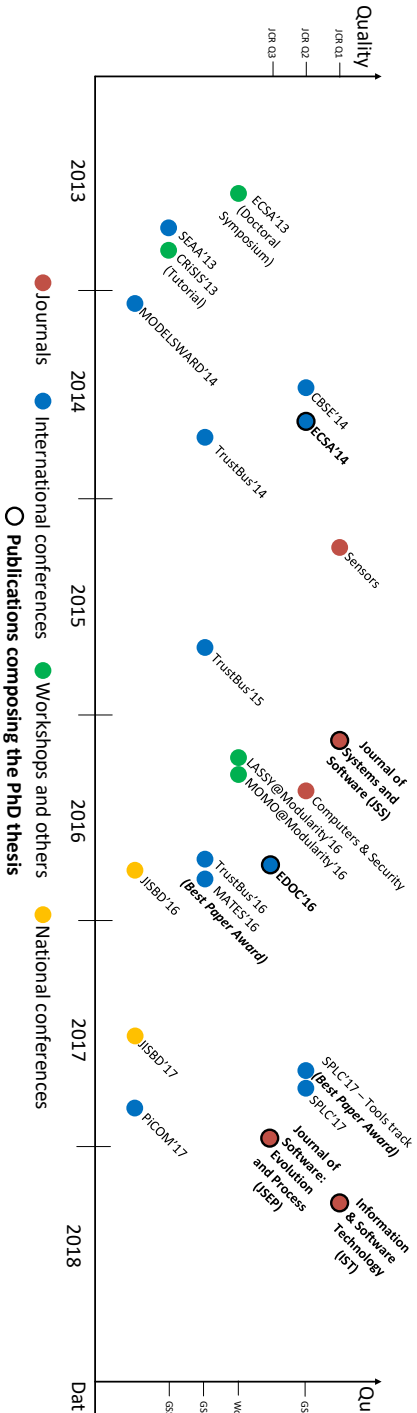

Figure A.2: Publications during the thesis.

Table A.1: Publications on journals.

JOURNALS	
Title:	Context-dependent reconfiguration of autonomous vehicles in mixed traffic
Authors:	José Miguel Horcas,* Julien Monteil, Mélanie Bouroche, Mónica Pinto, Lidia Fuentes, Siobhán Clarke
Journal:	Journal of Software: Evolution and Process (JSEP)
JCR Impact Factor:	1.033 (Q3)
Publication Date:	April 2018
DOI:	http://onlinelibrary.wiley.com/doi/10.1002/smr.1926/full
Title:	Variability models for generating efficient configurations of functional quality attributes
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Journal:	Information and Software Technology (IST)
JCR Impact Factor:	2.694 (Q1)
Publication Date:	March 2018
DOI:	https://doi.org/10.1016/j.infsof.2017.10.018
Title:	An approach for deploying and monitoring dynamic security policies
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes, Wissam Mallouli, Edgardo Montes de Oca
Journal:	Computers & Security (CS)
JCR Impact Factor:	2.849 (Q2)
Publication Date:	May 2016
DOI:	https://doi.org/10.1016/j.cose.2015.11.007
Title:	An automatic process for weaving functional quality attributes using a software product line approach
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Journal:	Journal of Systems and Software (JSS)
JCR Impact Factor:	2.444 (Q1)
Publication Date:	February 2016
DOI:	https://doi.org/10.1016/j.jss.2015.11.005
Title:	Dynamic reconfiguration of security policies in wireless sensor networks
Authors:	Mónica Pinto,* Nadia Gámez, Lidia Fuentes, Mercedes Amor, José Miguel Horcas, Inmaculada Ayala
Journal:	Sensors
JCR Impact Factor:	2.033 (Q1)
Publication Date:	March 2015
DOI:	https://doi.org/10.3390/s150305251

* Corresponding author.

Table A.2: Publications on international conferences.

INTERNATIONAL CONFERENCES (I)	
Title:	Self-adaptive energy-efficent applications: The HADAS developing approach
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes, Nadia Gámez
Conference:	The 15th IEEE International Conference on Pervasive Intelligence and Computing (PiCOM)
Conference Rating:	—
Publication Date:	November 2017
DOI:	https://riuma.uma.es/xmlui/handle/10630/14803
Title:	Extending the Common Variability Language (CVL) engine: A practical tool (***Hitachi young best paper award*** - Data, Demonstrations and Tools track)
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	The 21st International Systems and Software Product Line Conference (SPLC)
Conference Rating:	GGS: A- / LiveSHINE: A / MA: A-
Publication Date:	September 2017
DOI:	http://doi.acm.org/10.1145/3109729.3109749
Title:	Green configurations of functional quality attributes
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	The 21st International Systems and Software Product Line Conference (SPLC)
Conference Rating:	GGS: A- / LiveSHINE: A / MA: A-
Publication Date:	September 2017
DOI:	http://doi.acm.org/10.1145/3106195.3106205
Title:	Using models at runtime to adapt self-managed agents for the IoT (***Best paper award***)
Authors:	Inmaculada Ayala, José Miguel Horcas,* Mercedes Amor,* Lidia Fuentes
Conference:	14th German Conference on Multiagent System Technologies (MATES)
Conference Rating:	GGS: Work in Progress / CORE: B / LiveSHINE: C / MA: C
Publication Date:	September 2016
DOI:	https://doi.org/10.1007/978-3-319-45889-2_12
Title:	Product line architecture for automatic evolution of multi-tenant applications
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)
Conference Rating:	GGS: B / CORE: B / LiveSHINE: A- / MA: B
Publication Date:	September 2016
DOI:	https://doi.org/10.1109/EDOC.2016.7579384
Title:	Automatic Enforcement of Security Properties
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	13th International Conference On Trust, Privacy & Security In Digital Business (TrustBus)
Conference Rating:	GGS: Work in Progress / CORE: B / MA: C
Publication Date:	September 2016
DOI:	https://doi.org/10.1007/978-3-319-44341-6_2
Title:	Dynamic deployment and monitoring of security policies
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes, Wissam Mallouli, Edgardo Montes de Oca
Conference:	12th International Conference On Trust, Privacy & Security In Digital Business (TrustBus)
Conference Rating:	GGS: Work in Progress / CORE: B / MA: C
Publication Date:	September 2015
DOI:	https://doi.org/10.1007/978-3-319-22906-5_14
Title:	Closing the gap between the specification and enforcement of security policies
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	11th International Conference On Trust, Privacy & Security In Digital Business (TrustBus)
Conference Rating:	GGS: Work in Progress / CORE: B / MA: C
Publication Date:	September 2014
DOI:	https://doi.org/10.1007/978-3-319-09770-1_10

* Corresponding author.

Table A.3: Publications on international conferences (cont.).

INTERNATIONAL CONFERENCES (II)	
Title:	Runtime enforcement of dynamic security policies
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	8th European Conference on Software Architecture (ECSA)
Conference Rating:	GGS: B / CORE: A / LiveSHINE: B / MA: C
Publication Date:	August 2014
DOI:	https://doi.org/10.1007/978-3-319-09970-5_29
Title:	Injecting quality attributes into software architectures with the common variability language
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	The 17th International ACM Sigsoft Symposium on Component-Based Software Engineering (CBSE)
Conference Rating:	GGS: B / CORE: A / SHINE: B / MAS: B-
Publication Date:	June 2014
DOI:	http://doi.acm.org/10.1145/2602458.2602460
Title:	An aspect-oriented model transformation to weave security using CVL
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)
Conference Rating:	GGS: Work in Progress / MA: C
Publication Date:	January 2014
DOI:	https://doi.org/10.5220/0004890601380147
Title:	Variability and dependency modeling of quality attributes
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)
Conference Rating:	GGS: B- / CORE: C / SHINE: B / MAS: B
Publication Date:	September 2013
DOI:	https://doi.org/10.1109/SEAA.2013.20

* Corresponding author.

Table A.4: Publications on workshops, tutorials, and doctoral symposium.

WORKSHOPS, TUTORIALS, AND DOCTORAL SYMPOSIUM	
Title:	Towards the dynamic reconfiguration of quality attributes
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	The 1st Workshop on Live Adaptation of Software SYstems (LASSY) - 15th International Conference on Modularity (Modularity)
Conference Rating:	GGS: A / CORE: A / LiveSHINE: A+ / MA: A
Publication Date:	March 2016
DOI:	http://doi.acm.org/10.1145/2892664.2892686
Title:	Towards contractual interfaces for reusable functional quality attribute operationalisations
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes, Steffen Zschaler
Conference:	The 1st International Modularity in Modelling Workshop (MOMO) - 15th International Conference on Modularity (Modularity)
Conference Rating:	GGS: A / CORE: A / LiveSHINE: A+ / MA: A
Publication Date:	March 2016
DOI:	http://doi.acm.org/10.1145/2892664.2892700
Title:	How to develop secure applications with aspect-oriented programming (***Tutorial***)
Authors:	Mónica Pinto,* José Miguel Horcas
Conference:	8th International Conference on Risks and Security of Internet and Systems (CRiSIS)
Conference Rating:	GGS: Work in Progress / CORE: C / MA: C
Publication Date:	October 2013
DOI:	https://doi.org/10.1109/CRiSIS.2013.6766345
Title:	Modeling of quality attributes using an aspect-oriented software-product line approach (***Doctoral Symposium***)
Authors:	José Miguel Horcas*
Conference:	7th European Conference on Software Architecture (ECSA)
Conference Rating:	GGS: B / CORE: A / LiveSHINE: B / MA: C
Publication Date:	July 2013
DOI:	https://info-web.lirmm.fr/ecoop13/images/ds/4-paper-jose%20miguel%20horcas%20aguilera.pdf

* Corresponding author.

Table A.5: Publications on national conferences.

NATIONAL CONFERENCES	
Title:	Configuración eco-eficiente de atributos de calidad funcionales
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	XXII Jornadas de Ingeniería del Software y Base de Datos (JISBD)
Conference Rating:	—
Publication Date:	July 2017
DOI:	https://riuma.uma.es/xmlui/bitstream/handle/10630/14389/JISBD2017_22.pdf?sequence=1&isAllowed=y
Title:	Evolución arquitectónica de servicios basada en modelos CVL con cardinalidad
Authors:	José Miguel Horcas,* Mónica Pinto, Lidia Fuentes
Conference:	XXI Jornadas de Ingeniería del Software y Base de Datos (JISBD)
Conference Rating:	—
Publication Date:	September 2016
DOI:	https://riuma.uma.es/xmlui/bitstream/handle/10630/12218/jisbd.pdf?sequence=1

* Corresponding author.

Appendix B

Resumen en Español

B.1 Introducción

Un *atributo de calidad* (QA) es una característica que afecta la calidad de los sistemas software [2]. La calidad de un sistema software se mide por el grado en el que posee una combinación de atributos de calidad como por ejemplo usabilidad, seguridad, rendimiento, eficiencia energética, persistencia, confidencialidad, confiabilidad y escalabilidad. Los QAs pueden afectar el diseño, el comportamiento en tiempo de ejecución y la experiencia del usuario de un sistema software. Por lo que deben ser bien identificados y modelados desde las primeras etapas del proceso de desarrollo. De hecho, los QA juegan un papel crítico en la fase de obtención de la arquitectura, sirviendo como un criterio de selección para elegir entre una gran cantidad de diseños alternativos, patrones e implementaciones finales.

Los QAs son normalmente tratados como *requisitos no funcionales* (NFR) que el sistema debe cumplir [3]. En algunos dominios, el cumplimiento de los QAs es aún más importante que el cumplimiento de los requisitos funcionales. Por ejemplo, en sistemas integrados, el tiempo de ejecución o la seguridad son requisitos críticos y su modelado tiene una complejidad aún mayor que el modelado de los requisitos funcionales. Aun así, existen algunos QAs que también describen comportamiento (e.g., seguridad, usabilidad) y deben tratarse del mismo modo que los requisitos funcionales [4]. Por ejemplo, para satisfacer seguridad, las aplicaciones deben incluir un componente de cifrado que modifique el comportamiento

del sistema para proporcionar confidencialidad. Esta tesis introduce el concepto de *Atributo de Calidad Funcional* (FQA) que se define como la funcionalidad específica que se incorpora en la aplicación para cumplir con los QA deseados, ya que describen el comportamiento funcional necesario para satisfacer algunos QAs específicos [6]. Ejemplos de FQAs son encriptación, hash, autenticación, ayuda contextual, almacenamiento, y logging. Por el contrario, en esta tesis, utilizamos el término *QAs no funcionales* o *propiedades no funcionales* (NFP) para referirnos a aquellos QAs relacionados con requisitos no funcionales que no pueden correlacionarse directamente con componentes software, pero sí que afectan a decisiones de diseño a nivel de arquitectura o de implementación. Ejemplos de QAs no funcionales son rendimiento, eficiencia energética, consumo de memoria, y coste.

Modelar estos FQAs es una tarea compleja por varias razones. Por un lado, se componen de muchas características relacionadas, por ejemplo *seguridad* está compuesto, entre otros, por *autenticación*, *confidencialidad* y *encriptación*. Tienen dependencias entre ellos que pueden pasar desapercibidas para el arquitecto software, por ejemplo, *seguridad* es requerido por otros atributos como *usabilidad* o *persistencia*. Por otro lado, tienen muchos puntos de variabilidad: una aplicación concreta puede requerir sólo *autenticación* y *control de acceso* mientras que otra aplicación puede necesitar sólo *encriptación*.

En resumen, hay mucha variabilidad en los FQAs y su modelado puede verse como una "familia de productos" en el sentido utilizado en los enfoques de Línea de Productos Software (SPL) [7]. En el contexto de los FQA, una ''configuración de un producto" es el subconjunto de característcas de los FQAs que satisface los requisitos de calidad de una aplicación en particular (por ejemplo, seguridad, persistencia, control de errores).

Por otro lado, los componentes de los FQAs que se incoporan a la aplicación suelen afectar a los QAs no funcionales, como el rendimiento y la eficiencia energética del sistema. Estos QAs no funcionales generalmente compiten y entran en conflicto entre sí. Por ejemplo, el uso de la implementación más energeticamente eficiente de un componente reduce el consumo de energía, pero puede penalizar el rendimiento del sistema. Por lo tanto, si un sistema quiere optimizar los QAs de eficiencia energética y rendimiento, se necesita llegar a un compromiso. Es muy importante investigar conjuntamente las relaciones entre los FQAs y los QAs no

funcionales. Encontrar la configuración óptima de los FQAs que satisfaga todos los requisitos de la aplicación y que, además, tenga en cuenta los QAs no funcionales es una tarea difícil y propensa a errores si se lleva a cabo manualmente.

Otra dificultad de modelar los FQA es que su funcionalidad suele afectar a varios componentes de la misma aplicación. Por lo que la funcionalidad de los FQAs suele estar mezclada y/o dispersa con la funcionalidad básica de la aplicación — esto es, la funcionalidad de los FQAs es transversal a la funcionalidad de las aplicaciones. Por ejemplo, la seguridad debe garantizarse generalmente en diferentes puntos de una aplicación. En particular, un FQA puede implicar la incorporación de varios componentes adicionales en diferentes lugares, como la encriptación que requiere un lugar donde cifrar los datos, y otro lugar donde descifrar los mismos datos, es decir, la encriptación está *dispersa* entre los diferentes componentes de la aplicación. Además, un componente puede necesitar ser seguro y persistente, por lo tanto, los FQAs de seguridad y persistencia están *mezclados* en el mismo componente de la aplicación base. Una tecnología software ampliamente utilizada para hacer frente a estos problemas es el Desarrollo de Software Orientado a Aspectos (AOSD) [8], donde las propiedades transversales se modelan en entidades separadas (los aspectos) que son luego "tejidos" o `integrados` con los componentes de la aplicación. Usando AOSD, los FQAs se pueden modelar y diseñar de forma separada a las aplicaciones, pueden personalizarse de acuerdo con los requisitos específicos de la aplicación e incorporarlos a la aplicación de una manera no invasiva. Esto es posible porque la definición de los FQA es independiente de las aplicaciones que los necesitan (por ejemplo, la implementación de un algoritmo de encriptación es independiente de la aplicación que lo utiliza). Modelar los FQAs por separado de la aplicación base tiene muchas ventajas: reutilización, arquitecturas menos acopladas, se facilita el mantenimiento y se mejora la evolución del software.

Finalmente, los FQAs pueden evolucionar en el futuro. En primer lugar, los requisitos de las aplicaciones pueden cambiar con el tiempo y esto implica actualizar las configuraciones previas de los FQAs incorporados en las aplicaciones. En segundo lugar, los frameworks y bibliotecas software que proporcionan implementaciones de los FQAs están evolucionando y actualizandose continuamente para ser competitivos en el mercado: nuevos métodos de autenticación (e.g., au-

tenticación con identificadores de redes sociales, biometría) o nuevas técnicas de persistencia (e.g., fragmentación de bases de datos, grupos de afinidad) aparecen con frecuencia. Esto implica que la "familia de FQAs" necesita actualizarse con las nuevas características, y por lo tanto, las configuraciones ya desplegadas de los FQAs deben actualizarse en consecuencia.

Los trabajos existentes [33, 34, 35, 36, 37, 38, 39, 31, 40, 41, 42] que tratan de modelar FQAs suelen hacerlo como parte de una Línea de Productos Software (SPL) mezclando el modelado de los FQAs con la funcionalidad básica de la aplicación. Por lo que no gestionan los FQAs explícitamente, no tienen en cuenta las dependencias entre ellos ni pueden razonar o analizar de manera independiente a la aplicación como afectan a otros atributos de calidad no funcionales como el rendimiento o el consumo energético. Además, la mayoría de estos enfoques [33, 34, 35, 36, 37, 38, 40] hacen el análisis de los QAs únicamente desde el punto de vista no funcional (coste, mantenimiento, rendimiento) del producto final de la SPL. Y solo algunos de ellos [39, 31, 41, 42] abordan el modelado de la variabilidad de la parte funcional de los QAs.

Para abordar todos los problemas mencionados anteriormente, en esta tesis proponemos WeaFQAs, un enfoque de línea de productos software orientada a aspectos (AO-SPL) para gestionar los FQAs. WeaFQAs permite: (1) modelar las similitudes y la variabilidad de los atributos de calidad funcionales desde las primeras etapas del proceso de desarrollo, (2) gestionar las dependencias existentes entre los FQAs, (3) independizar el modelado de los FQAs de la arquitectura de la aplicación afectada, (4) configurar los FQAs en base a los requisitos de cada aplicación teniendo además en cuenta propiedades no funcionales como el rendimiento y el consumo energético de cada solución, (5) incorporar las configuraciones de los FQAs a la arquitectura de la aplicación de manera automática y sin tener que modificar manualmente los componentes existentes; y (6) gestionar la evolución de los FQAs cuando los requisitos cambien en el futuro. Para ello, se han combinado los beneficios proporcionados por varias tecnologías: líneas de productos software (SPL), desarrollo de software orientado a aspectos (AOSD) y desarrollo dirigido por modelos (MDD).

Se han realizado y comparado dos implementaciones de WeaFQAs usando diferentes lenguajes de variabilidad y de modelado, además de proporcionar soporte con

una herramienta basada en el lenguaje CVL (Common Variability Language) [21]. La propuesta se ha evaluado cualitativamente y cuantitativamente, y se ha aplicado a varios casos de estudios y aplicaciones reales como en el contexto de vehículos autonómos, políticas de seguridad, sistemas multi-agentes, y redes de sensores.

B.2 Propuesta: WeaFQAs

Esta tesis se centra en modelar y personalizar FQAs por separado de las aplicaciones, y luego incorporar las configuraciones de los FQAs en las aplicaciones combinando los beneficios proporcionados por varias tecnologías como líneas de producto software (SPL), desarrollo de software orientado a aspectos (AOSD) y desarrollo digiro por modelos (MDD). Proponemos un enfoque de Línea de Productos Software Orientada a Aspectos (AO-SPL), llamado *WeaFQAs*, que extiende el marco clásico de desarrollo de SPL [7] como se muestra en la Figura 5.1. WeaFQAs distingue los procesos de ingeniería del Dominio, Aplicación, Integración (Weaving) y Evolución, que se describen a continuación. Estos procesos de ingeniería de WeaFQAs se centran en el modelado y la gestión de FQAs, excluyendo el modelado de la funcionalidad básica de las aplicaciones. A pesar de esto, WeaFQAs necesita considerar los requisitos de la aplicación sobre los QAs y su arquitectura software para poder incorporar los FQAs.

En WeaFQAs se definen dos roles diferentes: (1) los *expertos del dominio*, que se encargan de definir los activos (artefactos) del proceso de ingeniería del dominio, y (2) los *ingenieros de la aplicación* o *arquitectos software*, que usan esos artefactos, reutilizándolos e instanciándolos para cada aplicación en los procesos de ingeniería de la aplicación y de integración. Los ingenieros de aplicaciones reutilizan el trabajo realizado por los expertos de dominio cuando usan la SPL para generar configuraciones específicas de acuerdo con los requisitos de la aplicación. En el proceso de ingeniería de evolución, tanto los expertos de dominio como los ingenieros de las aplicaciones toman parte. Mientras que los ingenieros de dominio se encargan de actualizar los artefactos reutilizables del proceso de ingeniería del dominio con los nuevos cambios, los ingenieros de las aplicaciones se encargan de proporcionar los nuevos requisitos del sistema para propagar automáticamente los cambios a los artefactos de los procesos de ingeniería de la aplicación y de

integración.

B.2.1 El proceso de ingeniería del dominio para los FQAs

El proceso de ingeniería del dominio (parte superior de la Figura 5.1) gestiona la complejidad y la variabilidad de los FQAs desde las primeras etapas del ciclo de vida de desarrollo. Los objetivos de este proceso de ingeniería del dominio son: (i) caracterizar los QAs y los FQAs; (ii) definir las características comunes y la variabilidad de los FQAs, incluyendo sus dependencias, y construir soluciones reutilizables de los FQAs como son la arquitectura software de los FQAs y los patrones para incorporarlos en las aplicaciones; y (iii) analizar la influencia de las diferentes configuraciones de los FQAs en las propiedades no funcionales (NFP), es decir, analizar las interdependencias entre los FQAs y los NFPs.

Este proceso toma como entrada la información disponible sobre el dominio de los QAs, como artículos de investigación, encuestas o catálogos de QAs, y la implementaciones existente de los FQAs como framworks reales y bibliotecas de FQAs. A partir de esta entrada, los expertos en el dominio identifican la variabilidad de los FQAs, así como las dependencias entre ellos, los modelos de uso de los FQAs y las propiedades no funcionales. El modelo de uso de un FQA se define como el conjunto de variables que pueden tener un efecto positivo o negativo en las NFPs (e.g., eficiencia energética y rendimiento) así como los valores que cada variable puede tomar. Por ejemplo, la caracterización del FQA de encriptación incluye (i) identificar los diferentes algoritmos de encriptación disponibles en los frameworks de seguridad existentes, junto con sus variables y parámetros tales como la longitud de la clave, el modo de operación de cifrado de bloque y el relleno; (ii) las dependencias entre encriptación y otros FQAs (e.g., hash); y (iii) el modelo de uso que para el FQA de encriptación incluye el tamaño del objeto o archivo a encriptar o desencriptar, así como las variables y parámetros específicos de la funcionalidad de cifrado (e.g., el modo de cifrado de bloque) que pueden afectar a las NFPs (e.g., rendimiento).

La caracterización de los FQAs permite especificar y definir un modelo de variabilidad para los FQAs y construir una arquitectura software variable. Los patrones arquitectónicos reutilizables para incorporar los FQAs en las aplicaciones

también son especificados por los expertos del dominio para definir cómo se deben componer los diferentes FQAs con la arquitectura base de la aplicación. Además, los modelos de los FQAs están enriquecidos con información sobre las NFPs, como la información sobre eficiencia energética y rendimiento recopilada a través del análisis empírico y/o experimentación de las diferentes configuraciones. Esta información se utilizará luego durante el proceso de ingeniería de la aplicación para generar configuraciones óptimas de los FQAs.

Un aspecto importante a destacar en WeaFQAs es que los artefactos del proceso de ingeniería del dominio serán completamente reutilizados por cualquier aplicación que desee configurar e incorporar FQAs en su arquitectura software. Esos artefactos son el modelo de variabilidad de los FQAs, la arquitectura software de los FQAs, los patrones para incorporar los FQAs en las aplicaciones, y la información de las NFPs recopiladas durante la experimentación.

B.2.2 El proceso de ingeniería de la aplicación para los FQAs

El proceso de ingeniería de la aplicación (parte central de la Figura 5.1) tiene como objetivo generar configuraciones de los FQAs que satisfagan los requisitos de una aplicación en particular. El objetivo del proceso de ingeniería de aplicaciones es vincular la variabilidad de los FQAs de acuerdo con los requisitos de la aplicación.

Para lograr este objetivo, el ingeniero de la aplicación identifica los QAs requeridos por la aplicación y crea una configuración de los FQAs que cumple esos requisitos. Esto se hace seleccionando las características que satisfacen los requisitos de los QAs de la aplicación, es decir, instanciando el modelo de variabilidad de los FQAs, incluidos los modelos de uso, previamente especificados durante el proceso de ingeniería del dominio. El ingeniero de la aplicación también selecciona una función objetivo (definida durante el proceso de ingeniería de dominio) para generar la configuración óptima de los FQAs (por ejemplo, aquella configuración que maximice la eficiencia energética). A menudo, el ingeniero de la aplicación solo puede proporcionar una instancia parcial del modelo de variabilidad y los modelos de uso, por lo que se genera una configuración parcial en estos casos. Tanto esta configuración parcial como la función objetivo se utilizan junto con la información

de las NFPs recopilada en el proceso de ingeniería del dominio para generar automáticamente una configuración completa y óptima de los FQAs. Esto ayuda al ingeniero de la aplicación a la hora de seleccionar entre varias configuraciones.

A partir del modelo de configuración completo, se genera automáticamente una configuración de la arquitectura de los FQAs con la variabilidad resuelta. La resolución de la variabilidad se realiza automáticamente con la herramienta `vEXgine` [82] que es un motor de ejecución personalizable y extensible para resolver la variabilidad, y está diseñado para dar soporte a WeaFQAs. El resultado del proceso de ingeniería de aplicaciones es una configuración de arquitectura de los FQAs que solo contiene los componentes de los FQAs que se necesitan de acuerdo con los requisitos de la aplicación.

B.2.3 El proceso de ingeniería de integración (weaving) de los FQAs

El proceso de ingeniería de integración (o weaving) de los FQAs a las aplicaciones (parte inferior de la Figura 5.1) se encarga de integrar la configuración de la arquitectura de los FQAs generada en el proceso anterior en la arquitectura base de la aplicación que se está construyendo. El objetivo de este proce es incorporar el modelo arquitectónico personalizado de los FQA en la aplicación base mediante la aplicación de los patrones de integración apropiados. En WeaFQAs, este proceso se basa en tecnologías orientadas a aspectos que permiten incorporar el modelo arquitectónico de los FQAs con la arquitectura software de la aplicación base de forma no intrusiva, es decir, sin modificar manualmente los componentes existentes en la arquitectura de la aplicación.

Este proceso de integración no es una tarea sencilla ya que cada FQA tiene que ser introducido en diferentes puntos de las aplicaciones. Además, cada FQA se incorporará según un patrón diferente, en función de la semántica de cada FQA. Por lo tanto, el ingeniero de la aplicación debe identificar los puntos en la aplicación donde se incorporarán los FQAs y asociar el conjunto de patrones de los FQAs con los componentes ya configurados de la arquitectura de los FQAs. Al igual que ocurría en el proceso de ingeniería de la aplicación para las configuraciones, el ingeniero puede proporcionar una instancia parcial de los patrones de

integración debido a la falta de información en los requisitos de la aplicación sobre los puntos exactos donde integrar los FQAs. En esos casos, WeaFQAs identificará automáticamente todos los puntos en la arquitectura de la aplicación donde sería posible aplicar el patrón del FQA específico. Con esa información, el ingeniero puede completar la instanciación de los patrones de integración.

Además, el proceso de integración se realiza automáticamente con la herramienta `vEXgine`, sin modificar manualmente la arquitectura base de la aplicación. El resultado de este proceso es la arquitectura software de la aplicación que también incorpora los FQAs requeridos.

B.2.4 El proceso de ingeniería de evolución de los FQAs

El proceso de ingeniería de evolución (parte derecha de la Figura 5.1) aborda la gestión de los FQAs cuando los requisitos de la aplicación cambian y/o la tecnología de los FQAs evoluciona. El objetivo de este proceso es actualizar los artefactos de los diferentes procesos de ingeniería de WeaFQAs y propagar los cambios a la aplicación final ya desplegada.

Las entradas de este proceso son: (i) los nuevos QAs que puedan aparecer en el futuro como, por ejemplo, la eficiencia energética que es uno de los QAs más recientes a tener en cuenta; (ii) las nuevas implementaciones de los FQAs o actualizaciones de los framework y bibliotecas existentes como, por ejemplo, una nueva versión del framework Spring; y (iii) los requisitos de los nuevos QAs de la aplicación que deben tenerse en cuenta. Los primeros dos tipos de entradas requieren que los expertos en el dominio actualicen manualmente los artefactos del proceso de ingeniería del dominio para incorporar los posibles nuevos productos a la SPL y ponerlos a disposición de las aplicaciones. La tercera entrada requiere que el ingeniero de la aplicación proporcione una nueva configuración con los nuevos requisitos.

En todos los casos, los cambios deben propagarse automáticamente a las configuraciones de los FQAs desplegadas previamente. Para automatizar esta tarea, WeaFQAs proporciona un conjunto de algoritmos de evolución para calcular la diferencia entre configuraciones y actualizar la configuración previa, pero también proporciona un algoritmo de evolución para propagar los cambios a la arquitectura

software de los FQAs, obteniendo como resultado, la aplicación final con los FQA evolucionados.

B.2.5 Implementación de WeaFQAs

Dado que la variabilidad se puede modelar usando varias técnicas, como modelos de características (*feature models*) [9], anotaciones [18, 19] o utilizando un lenguaje de variabilidad como CVL [21], en esta tesis se proporciona y comparan dos implementaciones diferentes de nuestro enfoque genérico utilizando diferentes lenguajes de variabilidad y de descripción de arquitectura:

1. Usando modelo de características (*feature models*) para especificar la variabilidad de los FQAs [9] y AO-ADL, un lenguaje de descripción de arquitectura orientada a aspectos [27], para modelar la arquitectura software de los FQAs. Esta implementación también utiliza un lenguaje de modelado de variabilidad (VML) [83] para vincular las características del modelo de variabilidad con los elementos de la arquitectura software de los FQAs.

2. Usando el lenguaje CVL [21] para especificar la variabilidad y los puntos de variación de los FQAs, y UML para modelar la arquitectura software de los FQAs.

Estas dos implementaciones de WeaFQAs se presentan y se comparan en detalle en el Capítulo 8.

B.2.6 vEXgine

`vEXgine`[1] [82] es la herramienta desarrollada en esta tesis para dar soporte a WeaFQAs. vEXgine es una implementación personalizable y extensible del motor de ejecución del lenguaje CVL [21]. vEXgine ha sido desarrollada debido a la falta de herramientas que dan soporte al enfoque CVL. La herramienta es totalmente compatible con el proceso de resolución de la variabilidad de CVL, incluido el mecanismo de delegación que se puede extender con diferentes motores de

[1] `http://caosd.lcc.uma.es/vexgine`

ejecución. Incluye una implementación del motor de delegación basado en transformaciones de Modelo a Modelo que utiliza un lenguaje de transformación de propósito general como ATL. La API Java proporcionada del motor de ejecución permite extender el enfoque CVL para satisfacer las necesidades industriales de modelado de variabilidad en SPLs.

B.3 Conclusiones y trabajo futuro

Este capítulo presenta las conclusiones de la tesis y el trabajo futuro.

B.3.1 Conclusiones

Nuestra investigación se ha centrado en definir un proceso automático siguiendo un enfoque de línea de productos software orientado a aspectos (AO-SPL) que persigue: (i) separar el modelado de los FQAs de las aplicaciones; (ii) generar FQAs adaptados a los requisitos de las aplicaciones optimizando las propiedades no funcionales del sistema; (iii) incorporar FQAs personalizados en las aplicaciones de una manera no intrusiva mediante la definición de patrones de integración orientados a aspectos; y (iv) dar soporte a la evolución de la aplicación guiada por los FQAs. Como resultado, se ha propuesto el enfoque WeaFQAs que combina tecnologías de SPLs, AOSD, y MDD. Se han desarrollado dos implementaciones de WeaFQAs, se ha evaluado cualitativemente y cuantitativamente, y se aplicado a múltiples casos de estudio.

Al separar el modelado de los FQAs de las aplicaciones base, nuestra propuesta mejora tanto la modularidad como la reutilización de los FQAs. AOSD tiene como objetivo lograr esta separación de los FQAs al considerarlos como propiedades transversales. La separación de los FQAs también facilita el modelado, el diseño y la implementación tanto de los FQAs como de las aplicaciones base. Por otro lado, una SPL ayuda a administrar los FQAs desde las primeras etapas del proceso de desarrollo, al facilitar el modelado de la variabilidad de los FQAs y el proceso de generación de las diferentes configuraciones de los FQAs personalizadas para las aplicaciones. El uso de una SPL mejora la capacidad de mantenimiento de las arquitecturas software. Además, nos hemos centrado en el lenguaje CVL, ya

que aumenta la escalabilidad y la reutilización de los modelos de variabilidad en comparación con los modelos de características tradicionales (*feature models*).

Esperamos que nuestro enfoque permita: (1) mejorar la modularización y la reutilización de los FQAs gracias a la tecnología AOSD aumentando la calidad de las aplicaciones; y (2) mejorar la capacidad de mantenimiento y la extensibilidad de los FQAs gracias a las SPLs.

B.3.2 Trabajo futuro

Como trabajo futuro, planeamos continuar investigando sobre el enfoque WeaFQAs para proporcionar soporte completo para la adaptación en tiempo de ejecución y la reconfiguración dinámica de los FQAs. Las aplicaciones que se ejecutan en entornos altamente dinámicos cambian continuamente sus requisitos en tiempo de ejecución. Si los nuevos requisitos afectan a los FQAs, éstos se deben adaptar dinámicamente. Ejemplos de estas aplicaciones pueden ser aplicaciones móviles que deben adaptarse a los cambios en su entorno. Por ejemplo, un usuario que se mueve de un entorno seguro a uno no seguro y se debe incorporar encriptación en sus aplicaciones móviles para cifrar las comunicaciones y hacerlas más seguras. Por lo tanto, planeamos dotar los FQAs y las aplicaciones con capacidades de adaptación dinámica y autogestión guiadas por los FQAs. Para ello se utilizará la tecnología y los beneficios de líneas de producto software dinámicas (DSPL) [93] como por ejemplo, haciendo que los modelos de los FQAs estén disponibles en tiempo de ejecución para permitir su personalización dinámica.

Además, también pensamos extender WeaFQAs para considerar otras propiedades no funcionales de las diferentes configuraciones de los FQAs. A parte de la eficiencia energética y el rendimiento, consideraremos el consumo de memoria o atributos cualitativos, como el nivel de seguridad y el nivel de usabilidad. A lo largo de esto, una problemática que surge es la optimización de varios atributos de calidad simultáneamente, especialmente cuando múltiples FQAs y NFPs tienen interacciones entre ellos. Por lo tanto, también planeamos estudiar algunas técnicas multiobjetivo para la generación de las configuraciones de los FQAs. Por ejemplo, programación por objetivos, programación compromiso o algoritmos evolutivos.

Para algunos FQAs, generar todas las configuraciones y realizar todos los ex-

perimentos dentro de un tiempo razonable puede ser una tarea difícil de abordar, más aún cuando las configuraciones deben generarse en tiempo de ejecución. En esos casos, la escalabilidad podría ser un problema del enfoque WeaFQAs, por lo que debemos encontrar formas óptimas para generar y gestionar las configuraciones de los FQAs, y analizar las NFPs de esas configuraciones cuando se utilizan implementaciones de frameworks reales de los FQA.

Finalmente, esperamos aplicar el conocimiento adquirido de las tecnologías utilizadas (por ejemplo, AOSD, SPL, MDD, CVL, modelos de características,...) a otros campos de ingeniería del software, no solo en el dominio de los QAs.

References

[1] S. Apel, D. Batory, C. Kstner, and G. Saake, *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013. xv, 10, 12

[2] M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock, "Quality Attributes," Carnegie Mellon University, Software Engineering Institute, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-95-TR-021 ESC-TR-95-021, 1995. 3, 75

[3] L. Chen, M. A. Babar, and B. Nuseibeh, "Characterizing architecturally significant requirements," *IEEE Software*, vol. 30, no. 2, pp. 38–45, March 2013. 3, 75

[4] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional" requirements really non-functional?: An investigation of non-functional requirements in practice," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 832–842. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884788 3, 24, 75

[5] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012, vol. 5. 4, 23

[6] F. D. Rodríguez, S. T. Acuña, and N. J. Juzgado, "Reusable solutions for implementing usability functionalities," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 4, pp. 727–756, 2015.

[Online]. Available: https://doi.org/10.1142/S0218194015500084 4, 23, 24, 76

[7] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. 5, 9, 10, 37, 76, 79

[8] R. E. Filman, T. Elrad, S. Clarke, M. Aksit *et al.*, *Aspect-oriented software development.* Addison Wesley, 2004. 7, 14, 77

[9] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Technical Report CMU/SEI-90-TR-021, 1990. 11, 43, 84

[10] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Annals of Software Engineering*, vol. 5, no. 1, pp. 143–168, 1998. 11

[11] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in *Software Reuse, 1998. Proceedings. Fifth International Conference on*, 1998, pp. 76–85. 11

[12] K. Czarnecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications.* New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. 11

[13] M. Eriksson, J. Börstler, and K. Borg, "The PLUSS approach – domain modeling with features, use cases and use case realizations," in *Software Product Lines*, ser. Lecture Notes in Computer Science, H. Obbink and K. Pohl, Eds. Springer Berlin Heidelberg, 2005, vol. 3714, pp. 33–44. 11

[14] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," in *Software Product Lines*, ser. Lecture Notes in Computer Science, R. Nord, Ed. Springer Berlin Heidelberg, 2004, vol. 3154, pp. 266–283. 11

[15] ——, "Formalizing cardinality-based feature models and their specialization," in *Software Process: Improvement and Practice*, 2005, p. 2005. 11

[16] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending feature diagrams with UML multiplicities," 2002. 11

[17] M. Riebisch, "Towards a more precise definition of feature models," in *Modelling Variability for Object-Oriented Product Lines*, M. Riebisch, J. O. Coplien, and D. Streitferdt, Eds. Norderstedt: BookOnDemand Publ. Co, 2003, pp. 64–76. [Online]. Available: http://www.theoinf.tu-ilmenau.de/~riebisch/home/publ/06-riebisch.pdf 11

[18] M. Fontoura, W. Pree, and B. Rumpe, *The UML profile for framework architectures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. 12, 22, 43, 84

[19] H. Gomaa, "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, pp. 218–218. 12, 43, 84

[20] C. Hunsen, B. Zhang, J. Siegmund, C. Kästner, O. Leßenich, M. Becker, and S. Apel, "Preprocessor-based variability in open-source and industrial software systems: An empirical study," *Empirical Software Engineering*, vol. 21, no. 2, pp. 449–482, Apr 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9360-1 12

[21] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen, and A. Svendsen, "Adding standardized variability to domain specific languages," in *International Software Product Line Conference*, ser. SPLC, 2008, pp. 139–148. 12, 43, 79, 84

[22] CVL Submission Team, "Common Variability Language (CVL), OMG revised submission," http://www.omgwiki.org/variability/, 2012. 13

[23] B. Combemale, O. Barais, O. Alam, and J. Kienzle, "Using CVL to operationalize product line development with reusable aspect models," in *VARY@MoDELS'12: VARiability for You*. Innsbruck, Autriche: ACM,

2012, varyMDE (bilateral collaboration between Inria and Thales). [Online]. Available: http://hal.inria.fr/hal-00730274 14

[24] J. B. F. Filho, O. Barais, J. Le Noir, and J.-M. Jézéquel, "Customizing the common variability language semantics for your domain models," in *Proceedings of the VARiability for You Workshop: Variability Modeling Made Useful for Everyone*, ser. VARY '12. New York, NY, USA: ACM, 2012, pp. 3–8. [Online]. Available: http://doi.acm.org/10.1145/2425415.2425417 14

[25] P. Tarr, H. Ossher, W. Harrison, and S. M. Sutton, "N degrees of separation: multi-dimensional separation of concerns," in *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*, May 1999, pp. 107–119. 15

[26] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyväskylä, Finland, June 9-13, 1997, Proceedings*, 1997, pp. 220–242. [Online]. Available: https://doi.org/10.1007/BFb0053381 15

[27] M. Pinto, L. Fuentes, and J. M. Troya, "Specifying aspect-oriented architectures in AO-ADL ," *Information and Software Technology*, vol. 53, no. 11, pp. 1165–1182, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584911001005 17, 43, 84

[28] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management.* John Wiley & Sons, 2013. 18

[29] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, no. 1–2, pp. 31–39, 2008. 18, 28

[30] D. Kolovos, R. Paige, and F. Polack, "The epsilon transformation language," in *Theory and Practice of Model Transformations*, ser. Lecture Notes in Computer Science, A. Vallecillo, J. Gray, and A. Pierantonio, Eds.

Springer Berlin Heidelberg, 2008, vol. 5063, pp. 46–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69927-9_4 18, 29

[31] R. d. O. Cavalcanti, E. S. de Almeida, and S. R. Meira, "Extending the RiPLE-DE process with quality attribute variability realization," in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS. ACM, 2011, pp. 159–164. [Online]. Available: http://doi.acm.org/10.1145/2000259.2000286 20, 21, 25, 29, 78

[32] J. Kuusela and J. Savolainen, "Requirements engineering for product families," in *Proceedings of the 22Nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 61–69. [Online]. Available: http://doi.acm.org/10.1145/337180.337189 20, 21

[33] M. Matinlassi, E. Niemelä, and L. Dobrica, *Quality-driven Architecture Design and Quality Analysis Method: A Revolutionary Initiation Approach to a Product Line Architecture*, ser. VTT publications. Technical Research Centre of Finland, 2002. 20, 25, 29, 78

[34] H. Zhang, S. Jarzabek, and B. Yang, "Quality prediction and assessment for product lines," in *Advanced Information Systems Engineering*, ser. LNCS. Springer Berlin Heidelberg, 2003, vol. 2681, pp. 681–695. [Online]. Available: http://dx.doi.org/10.1007/3-540-45017-3_45 20, 21, 24, 29, 78

[35] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "COVAMOF: A framework for modeling variability in software product families," in *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, 2004, pp. 197–213. [Online]. Available: https://doi.org/10.1007/978-3-540-28630-1_12 20, 27, 29, 78

[36] B. González-Baixauli, J. C. S. d. P. Leite, and J. Mylopoulos, "Visual variability analysis for goal models," in *Proceedings of the Requirements Engineering Conference, 12th IEEE International*, ser. RE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 198–207. 20, 21, 29, 78

[37] S. Jarzabek, B. Yang, and S. Yoeun, "Addressing quality attributes in domain analysis for product lines," *Software, IEE Proceedings* -, vol. 153, no. 2, pp. 61–73, 2006. 20, 21, 24, 29, 78

[38] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Advanced Information Systems Engineering*, ser. LNCS. Springer Berlin Heidelberg, 2005, vol. 3520, pp. 491–503. [Online]. Available: http://dx.doi.org/10.1007/11431855_34 20, 21, 29, 78

[39] L. Etxeberria and G. Sagardui, "Evaluation of quality attribute variability in software product families," in *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ecbs 2008)*, March 2008, pp. 255–264. 20, 24, 29, 78

[40] G. Zhang, H. Ye, and Y. Lin, "Quality attribute modeling and quality aware product configuration in software product lines," *Software Quality Journal*, vol. 22, no. 3, pp. 365–401, Sep 2014. [Online]. Available: https://doi.org/10.1007/s11219-013-9197-z 20, 24, 26, 29, 78

[41] M. Schöttle, O. Alam, J. Kienzle, and G. Mussbacher, "On the modularization provided by concern-oriented reuse," in *Companion Proceedings of the 15th International Conference on Modularity*, ser. MODULARITY Companion 2016. New York, NY, USA: ACM, 2016, pp. 184–189. [Online]. Available: http://doi.acm.org/10.1145/2892664.2892697 20, 25, 29, 78

[42] O. Alam, J. Kienzle, and G. Mussbacher, "Concern-oriented software design," in *Model-Driven Engineering Languages and Systems*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 8107, pp. 604–621. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41533-3_37 20, 25, 29, 78

[43] L. Etxeberria, G. Sagardui, and L. Belategi, "Quality aware software product line engineering," *J. Braz. Comp. Soc.*, vol. 14, no. 1, pp. 57–69, 2008. [Online]. Available: https://doi.org/10.1590/S0104-65002008000100006 21

[44] B. González-Baixauli, M. A. Laguna, and Y. Crespo, "Product line requirements based on goals, features and use cases," in *International Workshop on*

Requirements Reuse in System Family Engineering (IWREQFAM), jun 2004, pp. 4–7, http://polaris.dit.upm.es/ 21

[45] R. Tawhid and D. Petriu, "Automatic derivation of a product performance model from a software product line model," in *15th International Software Product Line Conference*, ser. SPLC, 2011, pp. 80–89. 21

[46] ——, "Integrating performance analysis in the model driven development of software product lines," in *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, ser. MoDELS. Springer-Verlag, 2008, pp. 490–504. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87875-9_35 21

[47] Y. Zhuang, "The performance cost of software obfuscation for android applications," *Computers & Security*, 2017. 22

[48] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske, "Model-based performance analysis of software architectures under uncertainty," in *International ACM Sigsoft Conference on Quality of Software Architectures (QoSA)*, 2013, pp. 69–78. 22, 23

[49] A. Martens, H. Koziolek, L. Prechelt, and R. Reussner, "From monolithic to component-based performance evaluation of software architectures," *Empirical Software Engineering*, vol. 16, no. 5, pp. 587–622, 2011. 22, 23

[50] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3 – 22, 2009, special Issue: Software Performance - Modeling and Analysis. 22, 23

[51] E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet, "Extending software architecture views with an energy consumption perspective," *Computing*, vol. 99, no. 6, pp. 553–573, Jun 2017. [Online]. Available: https://doi.org/10.1007/s00607-016-0502-0 22, 23, 32

[52] E. Jagroep, G. Procaccianti, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet, "Energy efficiency on the product roadmap:

An empirical study across releases of a software product," *Journal of Software: Evolution and Process*, vol. 29, no. 2, 2017. [Online]. Available: https://doi.org/10.1002/smr.1852 22, 23

[53] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W. G. J. Halfond, and J. Clause, "How does code obfuscation impact energy usage?" *Journal of Software: Evolution and Process*, vol. 28, no. 7, pp. 565–588, 2016. 22, 23

[54] G. Kalaitzoglou, M. Bruntink, and J. Visser, "A practical model for evaluating the energy efficiency of software applications," in *ICT4S*, 2014. 22

[55] K. Grosskop and J. Visser, "Energy efficiency optimization of application software," *Advances in Computers*, vol. 88, pp. 199–241, 2013. 22

[56] C. Tomazzoli, M. Cristani, E. Karafili, and F. Olivieri, "Non-monotonic reasoning rules for energy efficiency," *Journal of Ambient Intelligence and Smart Environments*, vol. 9, no. 3, pp. 345–360, 2017. 22

[57] D. Kim, J.-Y. Choi, and J.-E. Hong, "Evaluating energy efficiency of internet of things software architecture based on reusable software components," *International Journal of Distributed Sensor Networks*, vol. 13, no. 1, p. 1550147716682738, 2017. 22, 23

[58] D. J. Munoz, M. Pinto, and L. Fuentes, "HADAS and web services: Eco-efficiency assistant and repository use case evaluation," in *International Conference in Energy and Sustainability in Small Developing Economies (ES2DE)*, 2017, pp. 1–6. 22

[59] K. Djemame, D. Armstrong, R. Kavanagh, A. Juan Ferrer, D. Garcia Perez, D. Antona, J.-C. Deprez, C. Ponsard, D. Ortiz, M. Macías Lloret *et al.*, "Energy efficiency embedded service lifecycle: Towards an energy efficient cloud computing architecture," in *International Conference on ICT for Sustainability*, 2014, pp. 1–6. 22

[60] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories

software energy consumption framework," in *Working Conference on Mining Software Repositories*, 2014, pp. 12–21. 22

[61] S. Hasan, Z. King, M. Hafiz, M. Sayagh, B. Adams, and A. Hindle, "Energy profiles of java collections classes," in *International Conference on Software Engineering (ICSE)*, 2016, pp. 225–236. 23

[62] C. Stier, A. Koziolek, H. Groenda, and R. H. Reussner, "Model-based energy efficiency analysis of software architectures," in *European Conference on Software Architecture (ECSA)*, 2015, pp. 221–238. 23

[63] D. Feitosa, R. Alders, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, "Investigating the effect of design patterns on energy consumption," *Journal of Software: Evolution and Process*, vol. 29, no. 2, pp. e1851–n/a, 2017. 23

[64] B. Ouni, H. B. Rekhissa, and C. Belleudy, "Inter-process communication energy estimation through aadl modeling," in *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, ser. SMACD, 2012, pp. 225–228. 23

[65] N. J. Juzgado, A. M. Moreno, and M. I. Sánchez Segura, "Guidelines for eliciting usability functionalities," *IEEE Trans. Software Eng.*, vol. 33, no. 11, pp. 744–758, 2007. [Online]. Available: https://doi.org/10.1109/TSE.2007.70741 23

[66] J. Sincero, W. Schröder-Preikschat, and O. Spinczyk, "Approaching non-functional properties of software product lines: Learning from products," in *17th Asia Pacific Software Engineering Conference, APSEC 2010, Sydney, Australia, November 30 - December 3, 2010*, 2010, pp. 147–155. [Online]. Available: https://doi.org/10.1109/APSEC.2010.26 24

[67] N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov, "Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption," *Information & Software Technology*, vol. 55, no. 3, pp. 491–507, 2013. [Online]. Available: https://doi.org/10.1016/j.infsof.2012.07.020 24, 26

[68] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake, "SPL conqueror: Toward optimization of non-functional properties in software product lines," *Software Quality Journal*, vol. 20, no. 3-4, pp. 487–517, 2012. [Online]. Available: https://doi.org/10.1007/s11219-011-9152-9 24

[69] X. Peng, S. Lee, and W. Zhao, "Feature-oriented nonfunctional requirement analysis for software product line," *J. Comput. Sci. Technol.*, vol. 24, no. 2, pp. 319–338, 2009. [Online]. Available: https://doi.org/10.1007/s11390-009-9227-2 24

[70] A. M. Alashqar, A. A. Elfetouh, and H. M. El-Bakry, "Analyzing preferences and interactions of software quality attributes using choquet integral approach," in *International Conference on Informatics and Systems (INFOS)*, 2016, pp. 298–303. 24, 25

[71] F. Pinciroli, "Improving software applications quality by considering the contribution relationship among quality attributes," *Procedia Computer Science*, vol. 83, pp. 970 – 975, 2016. 24, 25

[72] J. Kienzle, W. Al Abed, and J. Klein, "Aspect-oriented multi-view modeling," in *Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development*, ser. AOSD. ACM, 2009, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/1509239.1509252 26

[73] 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. 27

[74] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch, "Modeling dependencies in product families with COVAMOF," in *13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, ser. ECBS, 2006, pp. 9 pp.–307. 27

[75] A. Egyed and P. Grunbacher, "Identifying requirements conflicts and cooperation: how quality attributes and automated traceability can help," *Software, IEEE*, vol. 21, no. 6, pp. 50–58, 2004. 27

[76] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A quality-driven systematic approach for architecting distributed software applications," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 244–253. 27

[77] A. McVeigh, J. Kramer, and J. Magee, "Using resemblance to support component reuse and evolution," in *Proceedings of the 2006 conference on Specification and verification of component-based systems*, ser. SAVCBS '06. New York, NY, USA: ACM, 2006, pp. 49–56. [Online]. Available: http://doi.acm.org/10.1145/1181195.1181206 28

[78] J. Pérez and et al., "Plastic partial components: A solution to support variability in architectural components," in *WICSA/ECSA*, 2009, pp. 221–230. 28

[79] A. Haber and et al., "Hierarchical variability modeling for software architectures," in *SPLC*, 2011, pp. 150–159. 28

[80] M. Sijtema, "Introducing variability rules in atl for managing variability in MDE-based product lines," *Proc of MtATL*, vol. 10, pp. 39–49, 2010. 28

[81] R. de Oliveira Cavalcanti, E. S. de Almeida, and S. R. L. Meira, "Extending the riple-de process with quality attribute variability realization," in *7th International Conference on the Quality of Software Architectures, QoSA 2011 and 2nd International Symposium on Architecting Critical Systems, ISARCS 2011. Boulder, CO, USA, June 20-24, 2011, Proceedings*, 2011, pp. 159–164. [Online]. Available: http://doi.acm.org/10.1145/2000259.2000286 31

[82] J. M. Horcas, M. Pinto, and L. Fuentes, "Extending the common variability language (CVL) engine: A practical tool," in *Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017*, 2017, pp. 32–37. [Online]. Available: http://doi.acm.org/10.1145/3109729.3109749 41, 43, 46, 82, 84

[83] N. Loughran, P. Sánchez, A. Garcia, and L. Fuentes, "Language support for managing variability in architectural models," in *Software Composition*, ser. LNCS. Springer Berlin Heidelberg, 2008, vol. 4954, pp. 36–51. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78789-1_3 43, 84

[84] J. M. Horcas, M. Pinto, and L. Fuentes, "An automatic process for weaving functional quality attributes using a software product line approach," *Journal of Systems and Software*, vol. 112, pp. 78–95, 2016. [Online]. Available: https://doi.org/10.1016/j.jss.2015.11.005 45, 46, 47

[85] ——, "Variability models for generating efficient configurations of functional quality attributes," *Information and Software Technology*, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095058491730383X 46, 47

[86] ——, "Runtime enforcement of dynamic security policies," in *Software Architecture - 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings*, 2014, pp. 340–356. [Online]. Available: https://doi.org/10.1007/978-3-319-09970-5_29 46, 48

[87] ——, "Product line architecture for automatic evolution of multi-tenant applications," in *20th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2016, Vienna, Austria, September 5-9, 2016*, 2016, pp. 1–10. [Online]. Available: https://doi.org/10.1109/EDOC.2016.7579384 46, 47, 48

[88] J. M. Horcas, J. Monteil, M. Bouroche, M. Pinto, L. Fuentes, and S. Clarke, "Context-dependent reconfiguration of autonomous vehicles in mixed traffic," *Journal of Software: Evolution and Process*, pp. e1926–n/a, 2017, e1926 smr.1926. [Online]. Available: http://dx.doi.org/10.1002/smr.1926 47, 48

[89] J. M. Horcas, M. Pinto, L. Fuentes, W. Mallouli, and E. M. de Oca, "An approach for deploying and monitoring dynamic security policies," *Computers & Security*, vol. 58, pp. 20–38, 2016. [Online]. Available: https://doi.org/10.1016/j.cose.2015.11.007 47

[90] M. Pinto, N. Gámez, L. Fuentes, M. Amor, J. Horcas, and I. Ayala, "Dynamic reconfiguration of security policies in wireless sensor networks," *Sensors*, vol. 15, no. 3, pp. 5251–5280, 2015. [Online]. Available: https://doi.org/10.3390/s150305251 47

[91] I. Ayala, J. M. Horcas, M. Amor, and L. Fuentes, "Using models at runtime to adapt self-managed agents for the iot," in *Multiagent System Technologies - 14th German Conference, MATES 2016, Klagenfurt, Österreich, September 27-30, 2016. Proceedings*, 2016, pp. 155–173. [Online]. Available: https://doi.org/10.1007/978-3-319-45889-2_12 47

[92] S. Ayed, M. S. Idrees, N. Cuppens-Boulahia, F. Cuppens, M. Pinto, and L. Fuentes, "Security aspects: A framework for enforcement of security policies using AOP," in *Ninth International Conference on Signal-Image Technology & Internet-Based Systems, SITIS 2013, Kyoto, Japan, December 2-5, 2013*, 2013, pp. 301–308. [Online]. Available: https://doi.org/10.1109/SITIS.2013.57 48

[93] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, April 2008. 52, 86