Closing the Gap between the Specification and Enforcement of Security Policies

José-Miguel Horcas, Mónica Pinto, and Lidia Fuentes

CAOSD Group, Universidad de Málaga, Andalucía Tech, Spain
{horcas,pinto,lff}@lcc.uma.es

Abstract. Security policies are enforced through the deployment of certain security functionalities within the applications. Applications can have different levels of security and thus each security policy is enforced by different security functionalities. Thus, the secure deployment of an application is not an easy task, being more complicated due to the existing gap between the specification of a security policy and the deployment, inside the application, of the security functionalities that are required to enforce that security policy. The main goal of this paper is to close this gap. This is done by using the paradigms of Software Product Lines and Aspect-Oriented Programming in order to: (1) link the security policies with the security functionalities, (2) generate a configuration of the security functionalities that fit a security policy, and (3) weave the selected security functionalities into an application. We qualitatively evaluate our approach, and discuss its benefits using a case study.

Keywords: Security enforcement, security policy, aspect-oriented programming, software product lines.

1 Introduction

A security policy is a set of rules that regulate the nature and the context of actions that can be performed within a system according to specific roles (i.e. permissions, interdictions, obligations, availability, etc) to assure and enforce security [1]. The security policies have to be specified before being enforced. This specification can be based on different models, such as OrBAC [2], RBAC [3], MAC [4], etc. and describes the security properties that an application should meet. Once specified, a security policy is enforced through the deployment of certain security functionalities within the application. For instance, the security policy "the system has the *obligation* to use a digital certificate to authenticate the users that connect using a laptop" should be enforced by deploying, within the application, "an authentication module that supports authentication based on digital certificates". This module must be executed before the user connects to the application using a laptop. In order to make explicit this relationship between the security policies and the security functionalities that are needed to enforce them, the links between both should be specified.

This relationship is needed because, normally, the same application can be deployed with different security policies. This implies that a variable number of security functionalities will be used by the application, but not all of them

C. Eckert et al. (Eds.): TrustBus 2014, LNCS 8647, pp. 106–118, 2014.

simultaneously. For example, in this paper we use an e-voting case study where the administrator can create elections of different types (e.g. national elections, coorporative elections, social event elections, etc.). These elections are deployed with different security policies. For instance, in a national election, users must be authenticated by an X.509 digital certificate provided by a trusted certification authority before they are joined to the election. Votes must be encrypted and must preserve their integrity and authenticity. However, in a corporative election users must be authenticated by using a user/password mechanism, while in a social event election users do not need to authenticate. In other words, there are different levels of security depending on the kind of election, and thus, each security policy is enforced by different security functionalities.

All this means that the secure deployment of an application is not an easy and straightforward task. Moreover, this is complicated even further due to the existing gap between the specification and the enforcement of a security policy. This gap is generated by the lack of a well-defined approach that would automatically introduce into an application, the security functionalities that are required to enforce the security policy. The main goal of this paper is to close this gap. We do this by using two advanced software engineering approaches, Software Product Lines (SPLs) [5] and Aspect-Oriented Programming (AOP) [6].

On the one hand, we use SPLs to: (1) model the commonalities and variabilities of the security properties represented in the security policies, (2) link the security properties to the modules that implement the security functionalities, and (3) automatically generate a security configuration including only the security functionalities that are required to enforce a particular security policy. On the other hand, we use AOP to: (1) design and implement the security functionalities separately from the applications (implementing them as aspects), and (2) deploy a security configuration in an application without modifying the original application. This work has been done in the context of the European project Inter-operable Trust Assurance Infrastructure (INTER-TRUST) [1] that aims to develop a framework to support trustworthy applications that are adapted at runtime to enforce changing security policies.

The rest of the paper is organized as follow. Section 2 describes the SPL and AOP approaches, introducing the main terminology. Section 3 provides an overview of our approach, while Section 4 and Section 5 describe it in an example-driven way using the e-voting case study. Section 6 evaluates our approach. The remaining sections present the related work, conclusions and future work.

2 Background Information

This section introduces the SPL and AOP approaches, and the main terminology.

2.1 Software Product Lines

In software engineering we usually need to create and maintain applications that contain: (1) a collection of similar functionalities from a shared set of software assets (i.e. *commonalities*) and (2) a collection of variant functionalities (i.e. *variabilities*). These applications require software developers to build a base on

the application commonalities, and to efficiently express and manage the application variabilities, allowing the delivery of a wide variety of applications in a fast, comprehensive and consistent way. The enabling technology to do that is the Software Product Lines (SPLs) [5]. SPLs allow the specification and modeling of the commonalities and variabilities of applications in an abstract level, creating different *configurations* of those variabilities for the implemented functionalities, and generating final applications with the customized functionalities according to those configurations.

The security functionalities that need to be deployed inside an application are clearly variable. One the one hand, security is composed by many concerns, such as authentication, authorization, encryption, and privacy concerns, among others, which are regarded as configurable functionalities of security. For instance, there are many possible mechanisms to authenticate users (e.g. digital certificate, password based, biometric based), or there are a variable number of places within an application where communications need to be encrypted. On the other hand, an application will require different levels of security, based on the requirements specified in different security policies. So, different security configurations for the same application can be generated by modeling security using an SPL.

Between the methods, tools and techniques provided by SPLs to model variability with the guarantee of a formal basis, some of the most commonly used are feature models [7] in which the variability functionalities are specified in the abstract level by using tree-based diagrams that include optionals and mandatories features, multiplicity, cross-tree constraint, among other characteristics. In this paper, we will use the Common Variability Language (CVL) [8] that apart from providing the same characteristics of feature models, is a domain-independent language and allows modeling and resolving the variability over models defined in any language based on Meta-Object Facility (MOF) [9].

2.2 Aspect-Oriented Programming

In object-oriented programming and component-based software engineering, there are concerns of an application that are dispersed or replicated in multiple modules. This occurs even when the functionalities are well-encapsulated in a class or a component (e.g. an encryption component), because the rest of the modules requiring these concerns need to include implicit calls to them (e.g. all the components in the application need to call the encryption component in order to encrypt/decrypt their interactions). These kinds of properties are known as *crosscutting concerns*, and their direct incorporation into the main functionality of an application cause *scattered* (i.e. dispersion) and *tangled* (i.e. mixing) code.

Security is a well-known crosscutting concern in the aspect-oriented community [10,11]. Figure 1 shows an example of how several security functionalities crosscut the base components of our e-voting application. For instance, `Authentication` is required for voters and administrators, so this functionality is scattered within the `Voter Client` and `Admin Client` components. In addition, the code of the authentication functionality is tangled with these main components. `Integrity` and `Signature` functionalities are also tangled within the `Voting Ballot` component; while `Encryption` is also dispersed in several

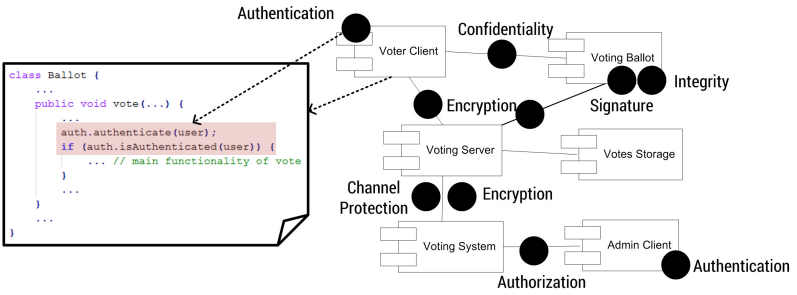

Fig. 1. Security functionalities crosscutting an e-voting application

places in the application. Modeling the variability of these security functionalities joinly with the base application is a difficult and error-prone task.

One of the most advanced techniques for dealing with crosscutting concerns is Aspect-Oriented Programming (AOP) [6]. AOP separates crosscutting concerns from the base functionality of an application by first encapsulating them in entities called *aspects*, and by then *weaving* (i.e. incorporating) these aspects into the existing code of the base application without modifying it. Aspects are described in terms of *join points*, *pointcuts* and *advices*. The points inside the code of the base application in which the crosscutting concerns come into play are known as *join points*; *pointcuts* are expressions that pick out certain join points; and *advices* is the crosscutting behaviour to be executed when a pointcut is reached. The mechanism in charge of weaving the main functionality of the application and the aspects is the *weaver*.

Separating the security functionalities from the base functionality of the application smoothes coupling between modules and increases the cohesion of each of them. As a consequence of a 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 thus, the modeling of the security variability and consequent deployment and enforcement of different security policies is easier. Moreover, the reusability also improves because the three elements (the base code, the security functionalities and the security policies) can be more easily reused in different systems.

3 Our Approach

As we presented in the introduction, an application can be deployed with different security policies and each security policy is enforced by different security functionalities. Top of Figure 2 shows the *problem* of the existing gap between the specification snd the enforcement of the security policies. Bottom of Figure 2 shows our *solution* to close this gap.

As previously mentioned, our approach combines the use of SPL and AOP. Firstly, as shown in Figure 2, left side under the `Solution` label, we model the variability of the security functionalities in an abstract level by specifying all the possible features of each security functionality in a tree-based diagram (i.e. we

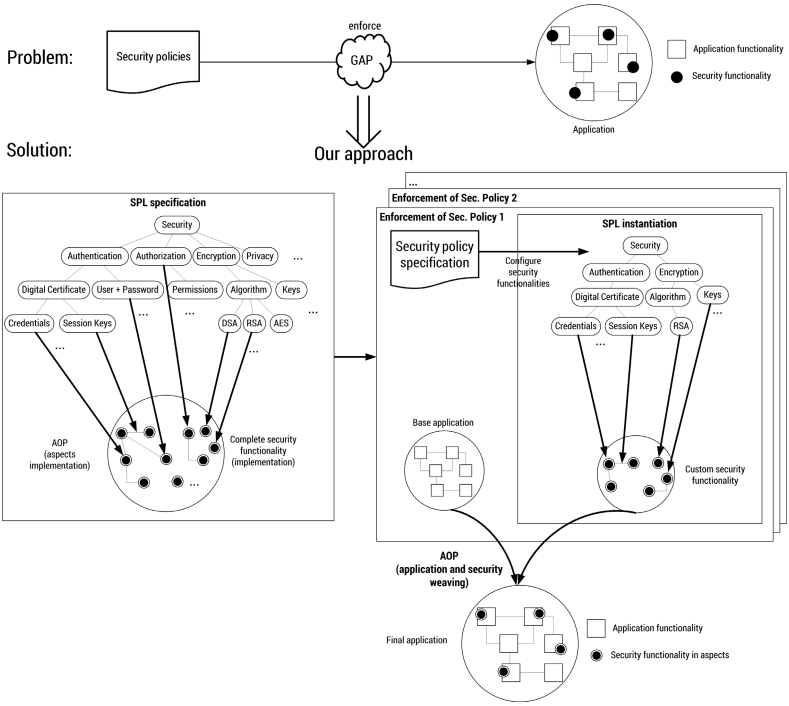

Fig. 2. Problem overview and solution proposal

specify the SPL). This variability model is linked to the complete implementation of the security functionalities, by linking each feature in the tree to the pieces of code that have to be included and configured (e.g. a class, a parameter, a value,...). Secondly, the right-hand side of Figure 2, under the `Solution` label, shows how, for each security policy, we create a configuration of the previously specified SPL by selecting those features in the tree that meet the security policy — i.e. we instantiate the SPL. This step can be done by a security expert or can also be done automatically by a reasoning engine that extracts the knowledge of the security policy and selects the appropriate features.

Using AOP, the security functionalities are implemented as aspects with the purpose of deploying them in the application without modifying the base code of the original application. Thus, the last step of our approach (see bottom half of Figure 2) is weaving the application and the aspects that implement a particular security configuration. The generated application includes the security functionalities that are needed to guarantee the enforcement of the security policy. The following sections describe our approach in more detail.

4 Variability Modeling of Security Functionalities

In this section we explain how to specify the SPL for modeling the variability of the security functionalities. We use the variability model of CVL that comprises two parts: (1) the variability specifications (VSpecs) tree (i.e. the abstract level)

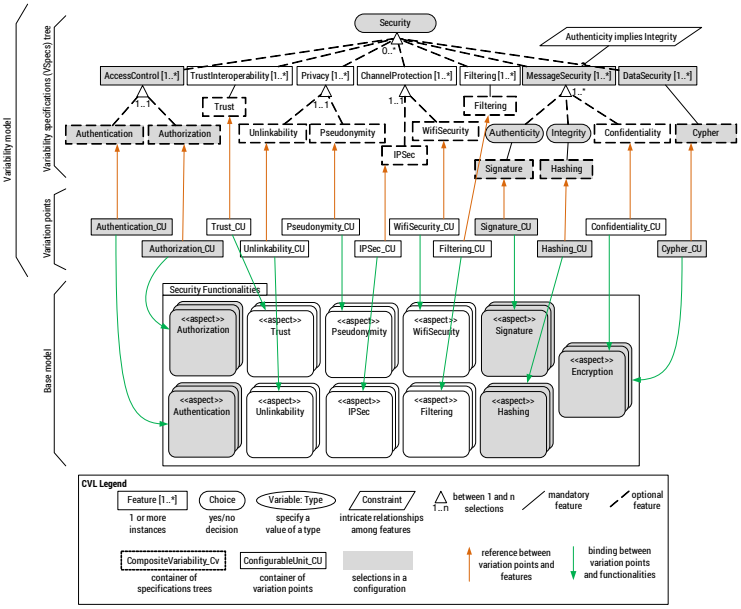

Fig. 3. Modeling security concepts in CVL

that allows us to specify the security features in a tree-structure with the relationships between them; and (2) the variation points, that allows us to link the features of the tree with the implementation of the security functionalities.

Figure 3 shows the variability model with these two parts and the implementation of the security functionality encapsulated inside the aspects. In the VSpecs, security is decomposed into more specific concepts and configurable features. For instance, the `AccessControl` concept contains the `Authentication` and `Authorization` features. The multiplicity [1..*] beside the `AccessControl` concept indicates that we can create multiple instances of it, each of them will require the configuration of different `Authentication` or `Authorization` features. For each of these instances there will be an aspect instantiated with the appropriate security functionality configured. In this case, variation points link each feature with the aspect that contains the configurable functionality.

Since security is composed by a lot of functionalities, and each of them has many configurable features, we define the variability model in two levels of detail using several related tree diagrams: (1) a high level VSpecs with all the main security functionalities represented as features (Figure 3); and (2) a VSpecs tree for each of these features in order to configure them appropriately. For instance, Figure 4 shows the part of the variability model that specifies the details to configure the `Authentication` functionality. Authentication is decomposed in a set of configurable features such as the authentication mechanism (`DigitalCertificate` or `UserPassword`) and the parameters and variables that contain the selected functionality (kind of certificate (`Credentials`), certificate authority (`TrustedCA`), ldots). These can be defined as optional features

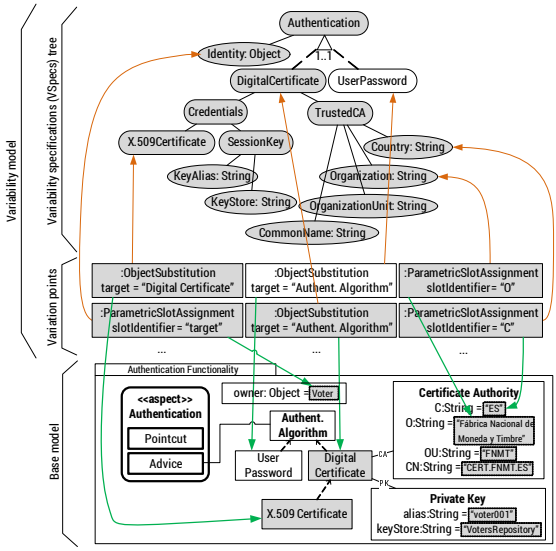

Fig. 4. Variability Modeling of Authentication functionality

(`ObjectExistence` variation point), variable features (`ObjectSubstitution` variation point), parameterizable features (`ParametricSlotAssignment` variation point), among others. In our case, we only need to use the `ObjectSubstitution` and the `ParametricSlotAssignment` variations point in order to substitute the selected authentication mechanism to be used (`Authent. Algorithm`) and to assign the values to the variables (e.g. parameters of the `TrustedCA`).

Note that the main benefit of our approach is that this SPL specification, although it is complex, only needs to be specified once by a software engineer, expert on security. Then, application developers will only have to select those security functionalities that need to be used in their applications and, as explained in the next section, our approach will generate a security configuration that enforces the required security policy, and is ready to be used by the application.

5 Enforcement of Security Policies

This section explains how to create a security configuration that enforces a security policy and how to deploy it within the application.

5.1 Configuring the security functionality

In order to select and configure the proper functionality that enforces a security policy, the security rules specified in the policy need to be analyzed and interpreted. This can be done by a security expert, an administrator, or automatically, by a reasoning engine that extracts the security knowledge from the rules, selects the security features, and assigns the appropriate values in the VSpecs — i.e. we instantiate the SPL by creating a configuration for the VSpecs.

Following our e-voting case study, Listing 1.1 shows an excerpt of a security policy defined in the OrBAC model for a national election in Spain. The first two rules specify that voters and administrators must be authenticated by a digital certificate in the spanish administration server (`GOB_ES`). The other rules specify permission for administrators and voters in order to create elections (rule 3) and to access the general elections (rule 4) respectively, to guarantee the authenticity and integrity of the votes (rule 5), and the encryption of all the interactions between the users and the server (rule 6).

Listing 1.1. Excerpt of an OrBAC security policy

```
1  Obligation(GOB_ES, User, Authenticate, Certificate, Authent_conditions) ∧
2  Obligation(GOB_ES, Admin, Authenticate, Certificate, Authent_conditions)
       ∧
3  Permission(GOB_ES, Admin, Create, Election, default) ∧
4  Permission(GOB_ES, User, Access, General_Election, default) ∧
5  Permission(GOB_ES, User, Sign, Votes, Signature_conditions) ∧
6  Permission(GOB_ES, User, Encrypt, Messages, Encrypt_conditions)
```

From these rules a configuration of the VSpecs is created. Some elements of the security policies are used to select the desired functionality, while other elements are used to configure the selected functionality. Features in a dark color in Figure 3 represent the security features selected for those rules. From rules 1-4 we have selected the `AccessControl` security concept with the `Authentication` (rules 1-2) and `Authorization` (rules 3-4) functionality, and this requires providing two different configurations for each of these functionalities — i.e. this implies two different instances of the authentication aspect and two different instances of the authorization aspect, properly configured. For rule 5 we have selected the `MessageSecurity` concept that includes the `Signature` functionality, but also the `Hashing` functionality since there is a constraint in the SPL specification indicating that `Authenticity implies Integrity`. So, in order to enforce this requirement two different aspects should be configured: `Signature` and `Hashing`. Finally, for rule 6 we have selected the `DataSecurity` concept with the `Cipher` functionality in order to encrypt the information exchange between the users and the server, and thus, an encryption aspect should be configured.

Each of these aspects contain the functionality that should be configured based on the knowledge specified in the policy. So, we have to provide a configuration for each instance of these aspects in the VSpecs. For example, features in a dark color in Figure 4 and concrete values assigned to the variables represent a configuration for the instance of the `Authentication` aspect corresponding to security rule 1 of the security policy[1]. For instance, the credential parameters and the trusted CA information are obtained from the context of the policy in the OrBAC model, but can also be manually assigned if the security model does not include that information.

In order to resolve the variability and automatically generate the configured security aspects, the CVL engine is executed taking as inputs the security variability model, the particular configuration created for the variability model, and the implementation of the security functionalities encapsulated into aspects.

[1] For reasons of space, we represent the configuration directly over the VSpecs.

5.2 Deploying the Security Aspects into the Application

Once the security aspects have been generated according the specifications of the security policy, the deployment of them inside the base application can be performed in a non-intrusive way —- i.e., without modifying the existing code of the application, by using the AOP. The mechanism in charge of composing the main functionality of the application and the aspects is the weaver, and how the weaver deploys the aspects within the application depends on the AOP framework chosen to define the aspects (e.g. AspectJ, Spring AOP, JBoss AOP, etc.). For instance, Figure 5 shows an example of the `Authentication` aspect (in AspectJ) woven with the e-voting application. This aspect encapsulates the authentication functionality (`AuthenticationModule`). The pointcut picks out the execution of the method `vote` and before it runs, the user is authenticated (advice code). We can observe how the application class `Ballot` does not contain any reference to the authentication functionality or to any other functionality related to security, which are all introduced as aspects.

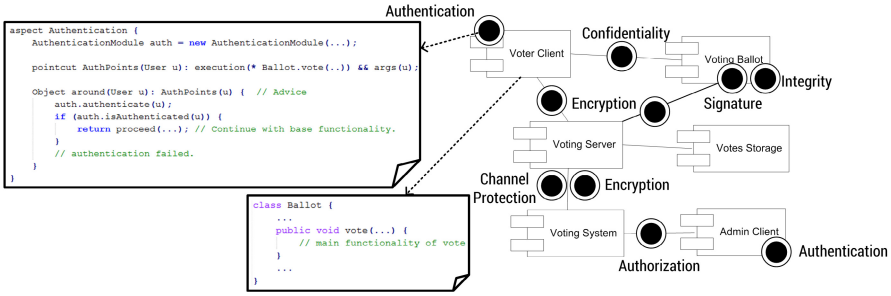

Fig. 5. Security functionality deployed within the e-voting application using AOP

6 Evaluation Results

Our approach uses consolidated software engineering technologies (SPLs and AOP), and a proposed standard language (CVL). So, in this section we qualitatively discuss our work to argue the correctness, maintainability, extendibility, separation of concerns, reusability, and scalability of our approach.

Correctness. SPLs and AOP do not improve the correctness of applications or security functionalities as such. Functionality in both cases is the same. However, modularizing security concerns in separate modules with AOP considerably facilitates the verification of the security properties of an application since a security expert does not have to check all the modules in the base application to ensure that all security requirements are correctly enforced. Instead, only the code of the aspects and the definition of the pointcuts where the aspects will be introduced need to be checked [10].

Maintainability and extendibility. On the one hand, due to the variability expressed in the SPL, changes of specifications in security policies are adapted easily by re-configuring the security functionality according to those changes.

Moreover, the variability model can be extended to cover more security concerns. On the other hand, AOP facilitates: (1) the modification of the security functionalities after being deployed due to the improved modularization, and (2) the extension of the points in which the security functionalities take place since we only need to extend the pointcuts of the aspects.

Separation of Concerns. Our approach improves the separation of concerns because we separate the specification of the security policies from the implementation of the core security functionalities, and from the deployment of the functionalities as aspects within the application.

Reusability. Following our approach, we can reuse the same security functionality with different applications and security policies. The main drawback is that we cannot reuse the same generated aspects for all the applications because the aspects are generated for a particular security policy and may also contain application dependent knowledge (e.g. pointcuts).

Scalability. Variability models have a considerable lack of scalability because the tree-diagrams become unmanageable as they grow (e.g. in feature models). However, CVL allows us to decrease the complexity of the model by dividing the VSpecs into different levels of details, using Composite Variability features and Configurable Units as we described in Section 4. Also, tree-diagrams is only syntactic sugar, but it is built onto a formal basis [7] and can also be specified using a text-based language [12].

In our follow-up work, we plan to improve our qualitative evaluation by using Delphi [13] techniques in order to evidence the benefits and usefulness of our approach from external experts.

7 Related Work

There is a growing interest in the SPL and AOP communities in resolving the gap between the security policies and the security functionalities that enforce the security policies [14,15,16]. For instance, in [14] the authors address the issue of formally validating the deployment of access control security policies. They use a theorem proving approach with a modeling language to allow the specification of the system jointly with the links between the system and the policy, and with the certain target security properties. The described algorithms, which perform the translation of the security policy inside specific devices' configurations, are based on the OrBAC model with the B-Method specifications, and thus, this approach does not close the gap completely, but only for specific OrBAC models. In addition, they do not separate the specification of the security functionality from the base functionality of the system as we do using AOP. In [17] the authors use a dynamic SPL to model runtime variability of services and propose different strategies for re-configuring a base model using CVL (e.g. model increments/decrements). They focus on the concrete variability transformations that need to be done in the base model in order to re-configured it, but they do not relate the specification of the requirements and the functionality provided as we do. Note that our approach can be extended to be applied to other requirements, not only for security.

The INTER-TRUST project [1] also aims to cover the gap, but in a different context. The INTER-TRUST framework regards the dynamic negotiation of changing security policies (specified in OrBAC) between two devices. The output of this negotiation is an agreement on interoperability policies, which must be dynamically enforced in both devices. This implies configuring applications and generating the security aspects that must be integrated into the applications [15]. The approach presented in this paper have to be seen as a possible implementation of the aspect generation module of the INTER-TRUST framework, which is the module in charge of determining the aspects that need to be deployed in order to enforce a security policy.

Several papers deal with security in an aspect-oriented way such as [11,16]. However, none of them consider security policies for the specification of the security requirements in the applications. Moreover, most approaches in the AOP community focus more on the security issues introduced by the AOP technology in the applications, such as in [10]. In [16], the authors propose a framework for specifying, deploying and testing access policies independently from the security model, but only suitable for Java applications. They follow a model-driven approach based on a generic security meta-model that is automatically transformed into security policies for the XACML platform by using the appropriate profile of the security model (e.g. OrBAC, RBAC). Then, the derived security components are integrated inside the applications using AOP. The main drawback to this framework is that the generated security components depend on the profiles of the specific security model used, and thus, the functionality cannot be reused. Moreover, re-deploying a security policy implies again generating the appropriate functionality with the consequent model transformations, which is a expensive process. The use of an SPL allows our approach the re-deployment of the security policy more easily and quickly.

In a previous work [18], we combined CVL and model transformations in order to weave security functionalities within the applications and focused on defining different weaving patterns for each security concern. Although the weaving process was inspired in AO modeling techniques, it was completely implemented in CVL without using AOP and the final application did not contain any aspects, in contrast to the approach presented in this paper where security is incorporated into the application using aspects. In [19] the authors also use CVL to specify and resolve the variability of a software design. However, their approach depends on an external Reusable Aspect Model (RAM) weaver to compose the chosen variants.

8 Conclusions and Future Work

The approach presented in this paper closes the existing gap between the specification and the enforcement of security policies by using consolidate software engineers technologies, such as SPLs and AOP. These techonologies bring significant benefits to our approach, including a better modularization, maintainability, extendibility, and reusability. Also the use of CVL as the variability modeling

language improves the scalability of our approach and makes it suitable for any MOF-based model. Moreover, separating the specification of the security policies from the implementation of the security functionalities, and from the deployment of them within the application, our approach is suitable for any security model and facilitates the verification of the security policies enforcement.

As part of our future work, we plan to adapt our approach to dynamically adapt the security functionalities to changes in the security policies at runtime. This implies using Dynamic SPLs [20] and generating the code of the aspects so as weaving them at runtime.

Acknowledgment. Work supported by the European INTER-TRUST FP7-317731 and the Spanish TIN2012-34840, FamiWare P09-TIC-5231, and MAGIC P12-TIC1814 projects.

References

1. INTER-TRUST Project: Interoperable Trust Assurance Infrastructure, `http://www.inter-trust.eu/`
2. Kalam, A., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: POLICY, pp. 120–131 (2003)
3. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur. 4(3), 224–274 (2001)
4. Sandhu, R.: Lattice-based access control models. Computer 26(11), 9–19 (1993)
5. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc. (2005)
6. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
7. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-021, Soft. Eng. Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania (1990)
8. Haugen, O., Wąsowski, A., Czarnecki, K.: CVL: Common Variability Language. In: SPLC, vol. 2, pp. 266–267. ACM (2012)
9. OMG: Meta Object Facility (MOF) Core Specification Version 2.0 (2006)
10. Win, B.D., Piessens, F., Joosen, W.: How secure is AOP and what can we do about it? In: SESS, pp. 27–34. ACM (2006)
11. Mouheb, D., Talhi, C., Nouh, M., Lima, V., Debbabi, M., Wang, L., Pourzandi, M.: Aspect-oriented modeling for representing and integrating security concerns in UML. In: Lee, R., Ormandjieva, O., Abran, A., Constantinides, C. (eds.) SERA 2010. SCI, vol. 296, pp. 197–213. Springer, Heidelberg (2010)
12. Classen, A., Boucher, Q., Heymans, P.: A text-based approach to feature modelling: Syntax and semantics of TVL. Science of Computer Programming 76(12), 1130–1143 (2011); Special Issue on Software Evolution, Adaptability and Variability
13. Gordon, T.J.: The delphi method. Futures Research Methodology 2 (1994)
14. Preda, S., Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J., Toutain, L.: Model-driven security policy deployment: Property oriented approach. In: Massacci, F., Wallach, D., Zannone, N. (eds.) ESSoS 2010. LNCS, vol. 5965, pp. 123–139. Springer, Heidelberg (2010)

15. Ayed, S., Idrees, M.S., Cuppens-Boulahia, N., Cuppens, F., Pinto, M., Fuentes, L.: Security aspects: A framework for enforcement of security policies using aop. In: SITIS, pp. 301–308 (2013)
16. Mouelhi, T., Fleurey, F., Baudry, B., Le Traon, Y.: A model-based framework for security policy specification, deployment and testing. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 537–552. Springer, Heidelberg (2008)
17. Cetina, C., Haugen, O., Zhang, X., Fleurey, F., Pelechano, V.: Strategies for variability transformation at run-time. In: SPLC, pp. 61–70 (2009)
18. Horcas, J.M., Pinto, M., Fuentes, L.: An aspect-oriented model transformation to weave security using CVL. In: MODELSWARD, pp. 138–147 (2014)
19. Combemale, B., Barais, O., Alam, O., Kienzle, J.: Using cvl to operationalize product line development with reusable aspect models. In: VARY, pp. 9–14 (2012)
20. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines. Computer 41(4), 93–95 (2008)