Runtime Enforcement of Dynamic Security Policies

Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes

Universidad de Málaga, Andalucía Tech, Spain
{horcas,pinto,lff}@lcc.uma.es
http://caosd.lcc.uma.es/

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.

Keywords: Security enforcement, Security policy, Aspect-Oriented Programming, Dynamicity.

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* of using 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".

However, the security policies of an application can change at runtime due to many 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 sense, the use

P. Avgeriou and U. Zdun (Eds.): ECSA 2014, LNCS 8627, pp. 340–356, 2014.

of the Autonomic Computing (AC) paradigm [5] is nowadays widely accepted by the distributed systems community to endow distributed systems with this dynamicity or self-management capacities.

Following the typical MAPE-K loop of the AC paradigm, where "MAPE" stands for Monitoring-Analysis-Plan-Execution and 'K' stands for Knowledge, the development of a software system with self-adaptation of the security functionalities consists on providing support to: (1) model the security information, including the identification of those features that are foreseen that may change at runtime and the mapping with the security functionalities (Knowledge); (2) model the security functionalities that need to be deployed in order to enforce a required security level (Knowledge); 3) monitor the runtime environment to listen for changes (e.g. contextual changes, user preferences changes, changes on the resources availability) that may affect security (Monitor); 4) analyze how the occurred changes affect the security configuration of the application (Analysis); 5) define a plan with the set of changes that need to be performed in the current security configuration (Plan Generation), and 6) dynamically adapt the security configuration according to the plan generated (Plan Execution).

In this paper we focus on presenting how the security knowledge can be modeled making use of a Dynamic Software Product Line (DSPL) [6] approach, and how the generation and execution of the reconfiguration plan can be developed using Aspect-Oriented Programming (AOP) [7]. On the one hand, DSPL are an accepted approach to manage the runtime (security) variability of applications. DSPLs produce software capable of adapting to changes, by means of binding the variation points at runtime [6]. This means that the variants of the DSPL are generated at runtime. On the other hand, the AOP technology is very appropriate to implement the dynamicity that is required in our approach. AOP produces more modular software with a better separation of concerns and this facilitates the runtime weaving and/or unweaving of the security functionality. The rest of the MAPE-K loop (i.e. the monitoring and analysis phases) are out of the scope of this paper.

Part of this work has been developed in the context of the FP7 European Project INTER-TRUST [1]. INTER-TRUST is a framework for the specification, negotiation, deployment and dynamic adaptation of inter-operable security policies. Concretely, the modules that perform the dynamic generation and execution of the security adaptation plan are also part of INTER-TRUST[1]. However, the use that we make of DSPLs to represent the security information and to generate the security configurations at runtime is specific of our approach.

After this introduction, the paper is organized as follow. Section 2 presents our proposal following the MAPE-K loop. Section 3 describes the knowledge, while Section 4 explains how to generate a new security configuration from the security policies. Section 5 and Section 6 describes the generation of the adaptation plan and the execution of the plan respectively. Section 7 evaluates our proposal.

[1] They are open source and can be downloaded from `https://github.com/Inter-Trust/Aspect_Generation/tree/demonstrator-version`

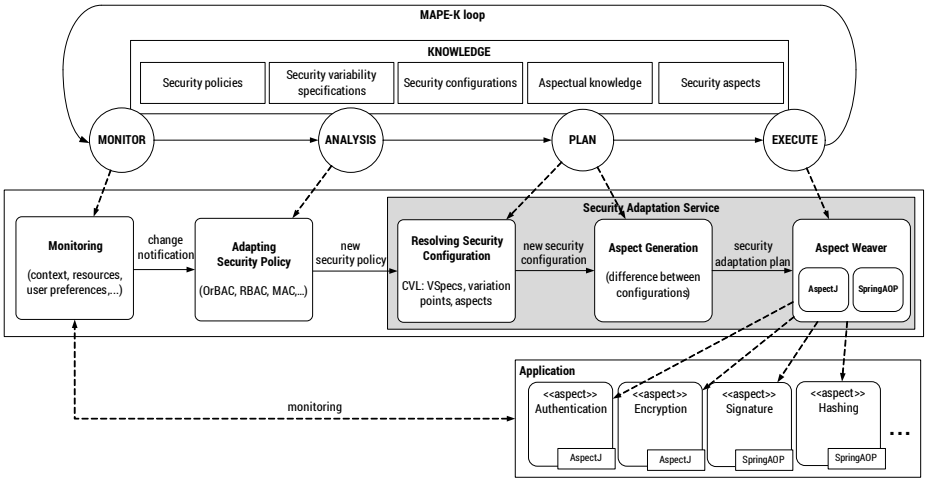

Fig. 1. Our MAPE-K loop approach and the Security Adaptation Service

Finally, Section 8 discusses the related work, and Section 9 presents conclusions and future work.

2 Our proposal

Figure 1 provides an overview of our proposal. As previously said, we follow the MAPE-K loop of the AC paradigm. An important part of the MAPE-K loop is the `Knowledge`, which in our approach represents all the information that is needed to adapt the applications to the changes on the security policies. The details about the `Knowledge` are provided in Section 3.

In the MAPE-K loop, the dynamic adaptations are driven by changes on the runtime environment. As shown in Figure 1, in our approach these changes are monitored and analyzed by the `Monitoring` and the `Adapting Security Policy` modules. These modules will depend on: which changes the application is interested on, such as changes in the context, changes in the user preferences, lack of enough resources in mobile environments, or negotiations of the security levels between interacting parties; the languages used to define the security policies (e.g. OrBAC [2], RBAC [3], MAC [4]), and the reasoning engine used to analyze and adapt those policies. The proposal presented in this paper is independent of a concrete design and/or implementation of these modules and thus the details about them are out of the scope of this paper. Basically, we will rely on existing approaches. For instance, in the context of the INTER-TRUST framework the monitoring is performed by the Montimage Monitoring Tool (MMT) [8] and the security policies are specified and analyzed using OrBAC [2].

Thus, this paper mainly focuses on the `Resolving Security Configuration`, the `Aspect Generation` and the `Aspect Weaver` modules that form the `Security Adaptation Service` (gray shaded in Figure 1). Firstly, the `Resolving Security`

`Configuration` module is in charge of selecting the proper configuration of the security functionality that needs to be deployed into the application in order to fulfill the requirements specified in the new security policy. All the possible security configurations are specified using a DSPL. Concretely, we use the Common Variability Language (CVL) [9] to specify and resolve the variability of the security functionalities. The details of this approach can be found in [10]. The main difference of both papers is that in [10] we used a SPL based on the use of CVL to generate a particular configuration of security during the design of applications. Now, we have extended that approach to use those security models at runtime and to integrate it in our `Security Adaptation Service`. These security functionalities are encapsulated into aspects by using AOP.

Secondly, the new security configuration is sent to the `Aspect Generation` module that dynamically generates a generic *security adaptation plan* with the actions that need to be performed with the security aspects (e.g. add a new aspect, remove an aspect, re-configure an aspect,...), taken into account the difference between the current security configuration deployed within the application and the new security configuration required, and using the available aspectual knowledge of the application (e.g. pointcuts and advices definitions).

Finally, the `Aspect Weaver` module executes the security adaptation plan at runtime by performing the particular actions of the AOP framework being used (e.g. weave/unweave). This module supports different AOP frameworks (AspectJ and Spring AOP) since the use of a unique AOP solution does not cover all the dynamicity, expressiveness, versatility, and performance requirements that the applications may need. For instance, Spring AOP allows weaving aspects at runtime (in contrast to AspectJ), and thus, we can add new security functionalities at runtime that were not taken into account when the application was initially deployed.

The Security Adaptation Service represents a generic solution that can be applied to many types of applications (e.g. pervasive applications, service-oriented applications, etc.) and can be used for the adaptation of any functionality, not only security. For instance, to illustrate our approach, we use an e-voting case study which is one of the demonstrators of the INTER-TRUST project where security requirements are complex and can change at runtime. This case study is provided by a enterprise partner of INTER-TRUST and a complete description of the e-voting application can be found in [1].

3 Knowledge

The knowledge represents all the information required in order to adapt the applications to changes on the security policies. The knowledge is available at runtime and includes: (1) the security policies that can be specified in any security model (e.g. OrBAC); (2) the security variability specifications; (3) the current security configuration deployed within the application; (4) the security aspectual knowledge; and (5) a repository with the security aspects files (i.e. class files, jar files, xml files,...).

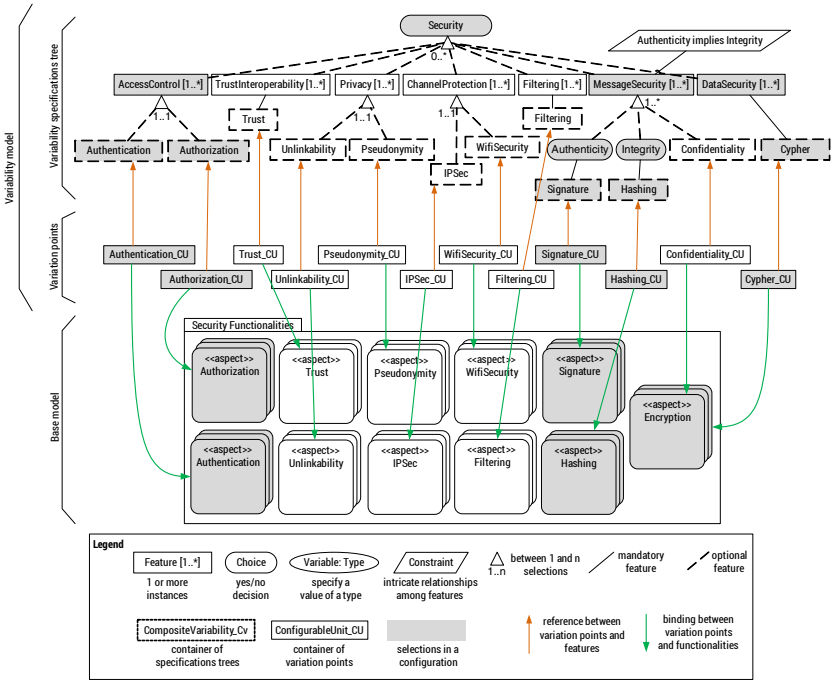

Fig. 2. Modeling security concepts in CVL

On the one hand, as previously said, the security knowledge is specified making use of DSPLs to manage the security variability at runtime. Figure 2 represents the security variability modeled in CVL. Top of Figure 2 shows the abstract part in which all the security concerns, functionalities, attributes and parameters that can be re-configured at runtime are specified — i.e. the variability specifications tree for security. Bottom of Figure 2 shows a representation of the particular implementation of the security functionalities encapsulated into aspects. Details of the parameters of each security concern in the variability specifications tree and in the aspects are hidden in Figure 2 to simplify it. Figure 3 shows a complete example for the authentication functionality and will be detailed in the next section. Finally, the security features of the tree and the specific functionality of the aspects are linked by using the CVL variation points[2] (middle of Figure 2). A particular selection of the features in the tree defines a particular configuration of the aspects. This configuration will be generated by the `Resolving Security Configuration` module, taking as input the requirements specified in the security policies.

On the other hand, the aspectual knowledge used by the `Aspect Generation` module depends on the application and on the implementation of the aspects since this information includes the points of the application where the security

[2] A complete description of CVL can be found in `http://www.omgwiki.org/variability/`

Listing 1.1. Security aspectual knowledge.

```
1  <ak:pointcuts>
2    <ak:pointcut id="Voter" expression="execution(* * Connection.*(
         Voter, ..)) && this(VoterClient)" />
3    <ak:pointcut id="sendingVoteJP" expression="execution(public *
         ElectionVote.sendVote(Vote, ..)) && args(Vote)" />
4  </ak:pointcuts>
5  <ak:advices>
6    <ak:advice id="certificateAuth" classname="uma.caosd.sas.
         Authentication"><ak:functionalities>
7      <ak:functionality id="authentication#digCertificate" />
8      <ak:functionality id="authentication#x509certificate" />
9    </ak:functionalities></ak:advice>
10   <ak:advice id="userPassAuth" classname="uma.caosd.sas.
         Authentication"><ak:functionalities>
11     <ak:functionality id="authentication#userPassword" />
12   </ak:functionalities></ak:advice>
13   <ak:advice id="encrypt" classname="uma.caosd.sas.Encryption"><
         ak:functionalities>
14     <ak:functionality id="confidentiality#encrypt" />
15     <ak:functionality id="confidentiality#rsa-encryption" />
16   </ak:functionalities></ak:advice>
17 </ak:advices>
18 <ak:advisors>
19     <ak:advisor id="certAuth" advice-ref="certificateAuth" pointcut-
           ref="Voter" />
20     <ak:advisor id="userPassAuth" advice-ref="userPassAuth" pointcut
           -ref="Voter" />
21     <ak:advisor id="encryptRSA" advice-ref="encrypt" pointcut-ref="
           sendingVoteJP" />
22 </ak:advisors>
```

functionality can be incorporated (i.e. the pointcuts definitions) and the list of functionalities provided by each aspect (i.e. the advices). The aspectual information is represented in a mapping table with the information needed to relate the different security functionalities required by the security configuration with the available advices implemented in the aspects. This information also includes the associations between the defined pointcuts and the advices — i.e. the advisors, following the Spring AOP terminology. The excerpt XML file in Listing 1.1 is an example of the aspectual knowledge for the e-voting application. We observe the lists of pointcuts (lines 2–7), the list of advices with the functionalities provided by each aspect (lines 9–38), and the list of advisors (lines 40–50).

4 Resolving Security Configurations

This section describes the generation a new security configuration that enforces the new security policy received from the `Adapting Security Policy` module.

When a request of a new security policy is received, the `Resolving Security Configuration` module extracts the security information from the rules of the security policy, selects the proper security features, and assigns the appropriate parameters in the security variability specifications tree of the DSPL. For instance, Figure 3 shows a particular configuration for the authentication functionality in the e-voting application, and thus the required configuration for the attributes and parameters of the authentication aspect. The configuration includes the use of an X.509 digital certificate as authentication mechanism; this

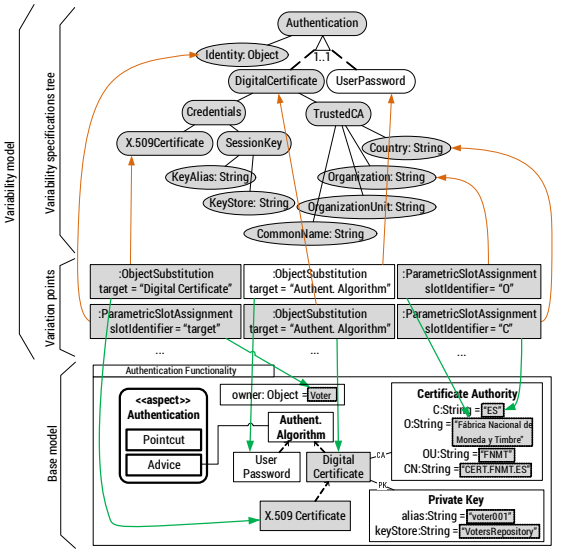

Fig. 3. Variability Modeling of Authentication functionality

Listing 1.2. New security configuration aspects.

```
<sca:aspects>
  <sca:aspect id="Authentication">
  <sca:joinpoint id="Voter" />
  <sca:functionalities>
    <sca:functionality id="authentication#digCertificate" />
    <sca:functionality id="authentication#x509certificate" />
  </sca:functionalities><sca:configuration>
    <sca:parameter name="SessionKey">
      KeyAlias="voter001", KeyStore="VotersRepository"
    </sca:parameter><sca:parameter name="TrustedCA">
      C="ES", O="Fabrica nacional de moneda y timbre", OU="FNMT", CN
          ="CERT.FNMT.ES"
    </sca:parameter></sca:configuration></sca:aspect>
</sca:aspects>
```

means that the authentication aspect must use an advice that implements an authentication algorithm based on digital certificates. Moreover, the configuration also includes the parameters for the certificate authority (`TrustedCA`) such as the information about the organization that issued the certificate, and the values of the session key to be used with the certificate (`KeyAlias` and `KeyStore`). Note that only one kind of authentication mechanism can be selected at the same time for the same instance of the aspect. But, the variability model allows creating and configuring different instances of each aspect.

Then, when the CVL engine is executed at runtime, it resolves the variability and automatically generates a configuration of the security aspects that enforces the new security policy. Listing 1.2 shows the new configuration generated for the authentication aspect. This new configuration must be deployed within the application, so this configuration is notified to the `Aspect Generation` module.

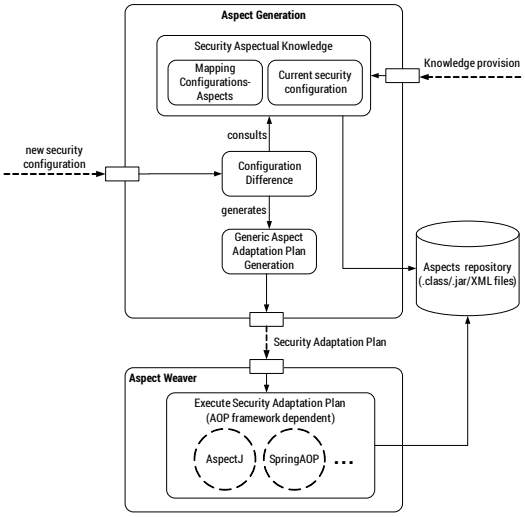

Fig. 4. Aspect Generation and Aspect Weaver architecture overview

Listing 1.3. Current security configuration deployed in the application.

```
<sca:aspects>
  <sca:aspect id="Authentication">
    <sca:joinpoint id="Voter" />
    <sca:functionalities>
    <sca:functionality id="authentication#userPassword" />
    </sca:functionalities></sca:aspect>
  <sca:aspect id="Privacy" ... />
  <sca:aspect id="Signature" ... />
  <sca:aspect id="Hashing" ... />
  <sca:aspect id="Pseudonymity" ... />
  <sca:aspect id="Unlinkability" ... />
</sca:aspects>
```

5 Aspect Generation

The `Aspect Generation` module receives notifications about a new security configuration to be deployed, and dynamically generates a generic security adaptation plan with the actions that need to be performed with the security aspects currently deployed in the application. This module is independent from the AOP framework used to weave the aspects. Its architecture and internal components are specified in top of Figure 4.

The `Security Aspectual Knowledge` component represents the part of the knowledge related to the security aspects (e.g. classnames, functionalities) and to the applications (e.g. pointcuts) that the `Aspect Generation` module requires (Listing 1.1), as well as the current security configuration of the aspects deployed in the application (Listing 1.3). The information is incorporated at the initialization of the module and can be updated at runtime, including the incorporation of new aspects (pointcuts and/or advices) to the aspect repository.

The `Configuration Difference` component analyses the notified new configuration and the aspectual knowledge and determines whether the security aspects, currently instantiated in the application, fulfil the new configuration or

Listing 1.4. Security adaptation plan

```
1 <sap:ADD>
2   <sap:advisor id="certAuth" />
3   <sap:advisor id="encryptRSA" />
4   <sap:advisor id="decryptRSA" />
5 </sap:ADD>
6 <sap:REMOVE>
7   <sap:advisor id="userPassAuth" />
8   <sap:advisor id="Pseudonymity" />
9   <sap:advisor id="Unlinkability" />
10 </sap:REMOVE>
11 <sap:CONFIGURE>
12   <sap:advisor id="certAuth" ><sap:configuration>
13     <sca:parameter name="SessionKey">
14       KeyAlias="voter001", KeyStore="VotersRepository"
15     </sca:parameter><sca:parameter name="TrustedCA">
16       C="ES", O="Fabrica nacional de moneda y timbre", OU="FNMT", CN
           ="CERT.FNMT.ES"
17     </sca:parameter></sap:configuration>
18   </sap:advisor>
19 </sap:CONFIGURE>
```

some changes must be done in the deployed aspects. To do that, we calculate the difference between the current security configuration deployed in the application and the new requested configuration. Then, using the aspectual knowledge and the security configuration calculated, the `Generic Aspect Adaptation Plan Generation` component generates a list of actions that need to be performed within the aspects in order to satisfy the security configuration calculated — i.e. generates the security adaptation plan. The list of actions are independent from the AOP framework and are based on the concept of advisor — i.e. the advice with the functionality and the associated pointcut defining the points of the application where the functionality takes place (see the aspectual knowledge in Listing 1.1). The possible actions are:

1. **ADD**. Deploys a new advisor within the application.
2. **REMOVE**. Undeploys an advisor currently deployed in the application.
3. **CONFIGURE**. Re-configures the parameters of an advisor currently deployed in the application or to be deployed.

For instance, as a result of the difference between the current security configuration deployed in our e-voting application (Listing 1.3) and the new configuration to be deployed (Listing 1.2), the list of actions to fulfill the calculated new configuration are presented in Listing 1.4. Since the authentication mechanism has changed, we need to remove the `userPassAuth` advisor and add a `certAuth` advisor, but we also need to configure the new `certAuth` advisor with the appropriate parameters (lines 16–28). Moreover, there are some other advisors to be added (`encryptRSA` and `decryptRSA`) and to be removed (`Pseudonymity` and `Unlinkability`) — see the selections of Figure 2. Advisors related to the `Privacy`, `Signature`, and `Hashing` aspects do not change, so no actions need to be performed for these three aspects.

The security adaptation plan is the input to the `Aspect Weaver` module described in the next section.

6 Aspect Weaver

The `Aspect Weaver` module receives a security adaptation plan and dynamically weaves, unweaves, and configures the security aspects at runtime interacting directly with them. Bottom of Figure 4 overviews the architecture of this module.

Since the security aspects can be implemented in more than one AOP framework in order to fulfill all the application needs, the `Aspect Weaver` module works as a wrapper that translates the generic security adaptation plan to the particular syntax of the AOP weaver being used (AspectJ, Spring AOP, etc). This means that different instantiations of the `Execute Security Adaptation Plan` component of this module for using different AOP weavers are available. The output of this component is a direct interaction with the selected AOP weaver in order to add, remove, and configure the corresponding aspects into the applications.

The specific actions to be performed depend on the dynamicity provided by each AOP weaver. On the one hand, the AspectJ weaver only supports compile-time and load-time weaving, while the Spring AOP weaver supports run-time weaving. This means that, in case of the AspectJ weaver, the security aspects need to be woven with the application at compile- or load-time weaving. However, we improve the dynamicity of our solution by using the `if()` pointcut constructor that AspectJ provides to define a conditional pointcut expression which will be evaluated at runtime for each candidate join point[3]. This mechanism increases the degree of dynamicity by coding patterns that can support dynamically enabling and disabling advice in aspects [11,12]. An example of the use of this mechanism to increase the dynamicity of the AspectJ aspects is shown in Listing 1.5. The `Authentication` aspect includes two advisors (`certificateAuth` and `userPassAuth`) that can be enabled or disabled at runtime by changing the advisor status. These advisors associate the pointcut with the proper advices defined in the aspect. The execution of each advice is based on the conditional pointcut to be evaluated at runtime. So, in this case, the action of adding (deploying) an advisor corresponds to enabling an advisor, and removing (undeploying) an advisor corresponds to disabling an advisor. This is done by the custom instance of the `Execute Security Adaptation Plan` component for AspectJ.

On the other hand, in the case of the Spring AOP aspects, the actions corresponding with the addition/deletion of an advisor are real operations allowed by the Spring AOP API[4] and are implemented following the proxy-based mechanism used by the Spring AOP framework to perform the run-time weaving. In this case, the instance of the `Execute Security Adaptation Plan` component for Spring AOP is in charge of managing all the Spring artifacts (e.g. advisors, proxies, XML configuration files,...) and performing the appropriate actions specified in the adaptation plan.

[3] `http://eclipse.org/aspectj/doc/released/progguide/index.html`
[4] `http://projects.spring.io/spring-framework/`

Listing 1.5. Authentication aspect in AspectJ.

```
1  public aspect Authentication {
2    ...
3    pointcut Voter(VoterClient v): execution(* * Connection.*(Voter,
         ..)) && this(v);
4    pointcut certificateAuth(VoterClient v): if(AdvisorsStatus.
         isEnabled("certificateAuth")) && Voter(v);
5    pointcut userPassAuth(VoterClient v): if(AdvisorsStatus.isEnabled(
         "userPassAuth")) && Voter(v);
6
7    Object around(VoterClient v): certificateAuth(v) {
8      CertificateAuthentication auth = new CertificateAuthentication(
           AdvisorsParameters.getParams("certificateAuth"));
9      if (auth.authenticate(v.getVoter()))
10       proceed();
11   }
12
13   Object around(VoterClient v): userPassAuth(v) {
14     UserPassAuthentication auth = new UserPassAuthentication(
           AdvisorsParameters.getParams("userPassAuth"));
15     if (auth.authenticate(v.getVoter()))
16       proceed();
17   }
18   ...
19 }
```

7 Evaluation

Our approach uses consolidated software engineering technologies (DSPLs and AOP), and a proposed standard language (CVL). So, in this section we first qualitatively discuss our work to argue about its correctness, maintainability, extensibility, separation of concerns, and reusability, from the point of view of the use of DSPLs and AOP. Regarding AOP, in spite of its benefits, the main concern about the use of this technology in real projects is the performance overhead introduced by AOP. This means that a critical part of the evaluation of our approach should be the evaluation of the performance overhead introduced by the use of a specific AOP weaver. As part of our participation in the INTER-TRUST project, the `Aspect Generation` and the `Aspect Weaver` modules presented in this paper has been used to implement a demonstrator of the project that provides dynamic adaptation of security for an e-voting application[5]. This demonstrator has been evaluated both quantitatively, by controlled tests performed for the implementation of the `Aspect Generation` and `Aspect Weaver` modules, and qualitatively, by collecting the opinion of software developers with different expertise on both security and AOP. The main results of this evaluation are discussed in this section.[6]

[5] The code and the documentation of this demonstrator can be downloaded from `https://github.com/Inter-Trust/Aspect_Generation/tree/demonstrator-version`

[6] For more detailed information the reader can consult the project deliverables [13,14].

7.1 Qualitatively Results

Correctness. DSPLs and AOP do not improve the correctness of applications or security functionalities as such. However, modularizing security functionalities 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. Additionally, it is well-known that the use of AOP can introduce vulnerabilities and security risks [15]. In our proposal, the `Monitoring` module is responsible for testing the behavior of the aspects [8] preventing these kinds of issues.

Separation of Concerns. The use of AOP improves modularization by allowing the separation of crosscutting concerns (i.e. the security functionalities in our approach). Moreover, following the MAPE-K loop we separate the different phases of our approach maintaining the independence of each module and facilitating the replacement of them.

Maintainability and extendibility. The use of DSPL allows us to easily reconfigure the security functionality according to the changes in the security policies. The variability model used (Figure 2) can also be extended to cover more security concerns.

Reusability. Our proposal is a generic solution that can be applied to many types of applications. The main drawback is that we cannot reuse completely the generated aspects for all the applications because they contain application dependent knowledge (e.g. pointcuts in the case of AspectJ). However, the security functionality (advices) can be reused in different contexts.

7.2 Performance overhead of AOP

AOP has important benefits in comparison to OO, such as achieving a better modularization of crosscutting concerns, improving the maintainability and the dynamic evolution of applications both at design and at runtime. These benefits are at the cost of a certain performance overhead, produced by the weaving process. In this evaluation, the main goal is to measure this performance overhead for the different AOP weavers and weaving mechanisms that we have used in the `Aspect Weaver` module, so we can reason about the suitability of using AOP for the `Security Adaptation Service`.

We have measured the time overhead introduced by the weaving process based on the lifetime of the application (compile-time, load-time, and run-time weaving) when the aspects are instantiated (Table 1) and when the advices of the aspects are executed (Table 2). We also consider the scalability of our solution when more than one aspect are applied at the same join point of the application. Results are summarized in Figure 5 and Figure 6. We observe that the overhead introduced by the AOP weavers is lower than the one initially expected. According to the data in Table 1, there is a penalty when the aspects are instantiated,

Table 1. Aspect weavers performance: aspects instantiation time (in milliseconds)

#aspect at the same join point:	1	5	10	20	50	100	1000
Compile-time weaving (AspectJ)	58.79	61.40	64.54	71.51	92.69	119.34	535.31
Load-time weaving (AspectJ)	19.81	20.91	30.11	53.72	109.46	196.22	1410.35
Run-time weaving (Spring AOP)	28.81	29.25	28.48	29.36	23.73	23.84	32.77

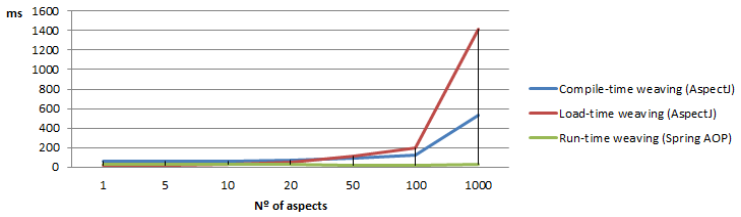

Fig. 5. Aspect weavers performance: aspects instantiation time

but, once the aspects have been created, the execution of them are faster (see Table 2). In any case, the results are similar for both the compile-time and load-time weaving. However, as expected, the runtime weaving introduces more overhead when the aspects are executed. Moreover, both AspectJ and Spring AOP weavers provide a great degree of scalability since a high number of aspects can be simultaneously applied at the same join point without reaching a non-acceptable performance overhead.

7.3 Results of the Software Developers Questionnaire

In [14] you can find a questionnaire about the usefulness of the INTER-TRUST framework in general, and in particular about the 'Aspect Generation and Aspect Weaver' demonstrator. That questionnaire was filled by evaluators that were selected mainly among software developers with different backgrounds and different levels of knowledge and experience in security issues and AOP.

Five participants who were experts in security modeling and negotiation answered the questions related to the Aspect Generation and Aspect Weaver demonstrator. In general, the results obtained are mainly in line with the expected target values (see [14] for more details about the metrics used to evaluate the demonstrator and the expected target values). However, some answers indicate that improvements can still be done for the next version of the demonstrator (and consequently, for the next version of the `Aspect Generation` and the `Aspect Weaver` modules). For instance, some evaluators were not convinced about the capacity of the `Aspect Generation` and `Aspect Weaver` modules to automatically deploy the security policies using aspects, its capacity for weaving the proper aspects or the runtime management of security policies and contextual information.

Additionally, five participants who were experts in security testing and monitoring also answered the questions related to the demonstrator. As for the experts on security modeling and negotiation, the results obtained were mainly in line with the expected target values.

Table 2. Aspect weavers performance: aspects execution time (in nanoseconds)

#aspect at the same join point:	**1**	**5**	**10**	**20**	**50**	**100**	**1000**
Compile-time weaving (AspectJ)	683	1280	1706	2560	5120	9813	106665
Load-time weaving (AspectJ)	426	854	1280	2560	4693	8960	111785
Run-time weaving (Spring AOP)	443301	439035	457808	474447	447568	479140	451408

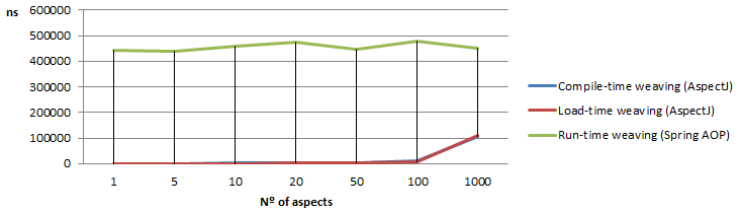

Fig. 6. Aspect weavers performance: aspects execution time

Finally, four participants who were experts in AOP answered the questionnaire. In general, we can say that the results obtained from AOP experts are better than the expected target values. Only one expert in AOP has considered that the security rules are not automatically deployed in the application original code. Since the rest of answers to this question are 'quite likely/extremely likely', we understand than the reviewer probably did not understand well either the question or how the modules functions regarding the automatic deployment of the aspects.

7.4 Discussion

The evaluation results obtained support our decision to use DSPLs and AOP in the design and implementation of our runtime `Security Adaptation Service`. However, we need to complete the evaluation with more interesting and conclusive empirical experiments. For instance, we need to evaluate the overhead of using AOP when different degrees of dynamicity are considered (e.g. when adding/removing advices and pointcuts) or when different instantiation models are used (e.g. aspect per object, aspect per control cflow,...). Moreover, we have not evaluated the global overhead introduced by the complete `Security Adaptation Service`, but only for the aspect solutions. Also, we need to increase the number of participants in the evaluation questionnaire in order to evidence the benefits and usefulness of our approach.

8 Related Work

There are a lot of works that try to deal with runtime adaptation of security. For instance, in [16] the authors present a policy-based approach for automating the integration of security mechanisms into Java-based business applications. They use security@runtime, an Domain Specific Language (DSL) for the specification of security configurations based on authorization, obligation and reaction policies. Our approach, in contrast, is suitable for using security policies specified in

any model (e.g. OrBAC), since the mapping between the policies and the security functionalities is made in an abstract level of the variability model. Another difference with our approach is that we separate the monitoring of changes in the application and the integration of the security functionality following the MAPE-K loop while they integrate the security functionalities at the same monitoring events. Moreover, they implement the security rules in separate classes but this code is application dependent while in our approach the security rules do not need to be hard-coded, improving the evolution of the policies.

In [17] the authors use policy-based security profiles for making logical and knowledge-based decisions within open service environments and it uses a layered holistic model [18] for describing security — i.e. security requirements are defined using security profiles that describe the interlinking of security policies to instances of services. However, in our approach, the security policies are decoupled from the specific knowledge of the application and from the implementation of the security functionality in aspects, and this improves the reusability of both the security policies and the security functionalities.

Model-Driven Security (MDS) are often used to adapt dynamically security following different approaches: UMLSec [19], SecureUML [20], OpenPMF [21,22], SECTET [23], etc. For instance, in [24], models@runtime is used to keep synchronized an architectural model with a policy, but this approach only supports access control policies and not any kind of security functionality as in our security adaptation service. This is a general limitation in many security policy based approaches because they are mainly focused only on access control or authorization concerns ([21,22,25,26]) or focused only on a specific domain such as mobile cloud ([27]) or Service Oriented Architecture (SOA) ([21,22,23,25]).

There are also some generic approaches for reconfiguration at runtime that are not focused only on security concerns. In [28], Gamez et al. propose a reconfiguration mechanism that switches among different architectural configurations at run-time. The configurations are based on the specialization of feature models, and the reconfiguration plans are automatically generated from the differences among them. They propagate changes in configurations at architectural level instead of directly aspects implementation, as we do.

9 Conclusions and Future Work

We have presented a complete solution for the run-time enforcement of security policies following the MAPE-K loop of the AC paradigm that endow multiple kinds of applications with this dynamicity and self-management capacities. We have described in detail a security adaptation service based on the combination of DSPL and AOP technologies, where the security functionalities are implemented as aspects that are dynamically configured, deployed or un-deployed by generating and executing a security adaptation plan. These technologies bring significant benefits to our proposal, including a better modularization, maintainability, extendibility, and reusability.

As part of our future work, we plan to complete the evaluation of our `Security Adaptation Service` with empirical studies in order to evidence its benefits and usefulness.

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. FP7 European Project INTER-TRUST: 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. IBM: Autonomic Computing White Paper - An architectural blueprint for autonomic computing. IBM Corp. (2005)
6. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines. Computer 41(4), 93–95 (2008)
7. 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)
8. Mallouli, W., de Oca, E.M., Wehbi, B., Fuentes, L., Pinto, M., Horcas, J.M., Benab, J.B., Prez, J.M.M., Ayed, S., Cuppens, N., Cuppens, F., Toumi, K., Cavalli, A., Kerezsi, E.: Specification and design of the secure interoperability framework and tools - first version. Deliverable D4.2.1, FP7 European Project INTER-TRUST (2013), `http://inter-trust.lcc.uma.es/documents/10180/15714/INTER-TRUST-T4.2-MI-DELV-D4.2.1-SpecDesSecInterFram`
9. Haugen, O., Wąsowski, A., Czarnecki, K.: CVL: Common Variability Language. In: SPLC 2012, vol. 2, pp. 266–267 (2012)
10. Horcas, J.M., Pinto, M., Fuentes, L.: Closing the gap between the specification and enforcement of security policies. In: TrustBus (2014)
11. Andrade, R., Ribeiro, M., Gasiunas, V., Satabin, L., Rebelo, H., Borba, P.: Assessing idioms for implementing features with flexible binding times. In: CSMR, pp. 231–240 (2011)
12. Andrade, R., Rebelo, H., Ribeiro, M., Borba, P.: Aspectj-based idioms for flexible feature binding. In: SBCARS, pp. 59–68 (2013)
13. Arrazola, J., Merle, L.: Specification of the evaluation criteria. Deliverable D5.2, FP7 European Project INTER-TRUST (2013), `http://inter-trust.lcc.uma.es/documents/10180/15714/INTER-TRUST+-++D5.2+Specification+of+the+evaluation+criteria/72c26aff-51fa-4117-b9ba-7afcac8468e0`

14. Bernab, J.B., Perez, J.M.M., Skarmeta, A.F., Pasini, R., Viszlai, E., Mallouli, W., Toumi, K., Ayed, S., Pinto, M., Fuentes, L., Horcas, J.M., Arrazola, J., Merle, L., Frontanta, J.L.V.: Results of first evaluation. Deliverable D5.3, FP7 European Project INTER-TRUST (2013), http://inter-trust.lcc.uma.es/documents/10180/15714/INTER-TRUST-T5.3-UMU-DELV-D5.3-ResultsFirstEval-V1.00.pdf/f8547c6e-bdbe-4be2-ade9-0698876d4423
15. 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)
16. Elrakaiby, Y., Amrani, M., Le Traon, Y.: Security@runtime: A flexible mde approach to enforce fine-grained security policies. In: Jürjens, J., Piessens, F., Bielova, N. (eds.) ESSoS. LNCS, vol. 8364, pp. 19–34. Springer, Heidelberg (2014)
17. Tan, J.J., Poslad, S.: Dynamic security reconfiguration for the semantic web. Engineering Applications of Artificial Intelligence 17(7), 783–797 (2004)
18. Tan, J.J., Poslad, S., Titkov, L.: A semantic approach to harmonizing security models for open services. Applied Artificial Intelligence 20(2-4), 353–379 (2006)
19. Jrjens, J.: Secure Systems Development with UML. Springer (2010)
20. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. 15(1), 39–91 (2006)
21. Lang, U.: OpenPMF SCaaS: Authorization as a service for cloud amp; SOA applications. In: CloudCom, pp. 634–643 (2010)
22. Lang, U.: Cloud & SOA application security as a service. In: ISSE 2010 Securing Electronic Business Processes, pp. 61–71 (2011)
23. Katt, B., Gander, M., Breu, R., Felderer, M.: Enhancing model driven security through pattern refinement techniques. In: Beckert, B., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 169–183. Springer, Heidelberg (2012)
24. Morin, B., Mouelhi, T., Fleurey, F., Traon, Y.L., Barais, O., Jézéquel, J.M.: Security-driven model-based dynamic adaptation. In: ASE, pp. 205–214 (2010)
25. Dong, W.: Dynamic reconfiguration method for web service based on policy. In: Electronic Commerce and Security, 61–65 (2008)
26. Gheorghe, G., Crispo, B., Carbone, R., Desmet, L., Joosen, W.: Deploy, adjust and readjust: Supporting dynamic reconfiguration of policy enforcement. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 350–369. Springer, Heidelberg (2011)
27. Cho, H.S., Hwang, S.M.: Mobile cloud policy decision management for mds. In: Lee, G., Howard, D., Kang, J.J., Ślęzak, D. (eds.) ICHIT 2012. LNCS, vol. 7425, pp. 645–649. Springer, Heidelberg (2012)
28. Gamez, N., Fuentes, L.: Software product line evolution with cardinality-based feature models. In: Schmid, K. (ed.) ICSR 2011. LNCS, vol. 6727, pp. 102–118. Springer, Heidelberg (2011)