An Approach to the Specification of Software Security Patterns

Spyros HALKIDIS, Alexander CHATZIGEORGIOU, George STEPHANIDES

Department of Applied Informatics
University of Macedonia, Thessaloniki, Greece
{halkidis, achat, steph}@uom.gr

ABSTRACT

Software Security patterns enforce security characteristics already at the design phase of a software system. They have been defined in analogy to the well-established Design Patterns that help to develop well-structured software. Since there is no systematic way to identify them we attempt to establish a common specification methodology, in order to enable an automatic recognition of existing security patterns within an object-oriented system

Keywords: coding theory, cryptography, Hierarchies
Math. Subjects Classification 2000: 94A60, 14G50, 68Q99

1. INTRODUCTION

Information systems security has been an active research area since decades [6, 13]. The importance of information systems security techniques has been eminent since the wide spread of computer communication technologies and the Internet. The focus was placed on network security, and network architecture techniques have been developed. Though, only recently it has been recognized that the main vulnerability that attacks questioning the security characteristics of information systems take advantage of, is in most cases software poorly designed and developed. Specifically, designed and developed without security being in the minds of people involved [18, 9, 16]. Through practical examples from attacks to businesses and universities it can be shown that security related attacks exploit in most cases so-called software holes. Taking this into account, a new field of research called software security has emerged during the last years [18,9]. In analogy to design patterns for building well-structured software [7], architectural patterns for building secure systems have been proposed. These patterns, called security patterns, have been an active research area since the work by Yoder and Barcalow [20]. Taking into account that a qualitative evaluation of security patterns [8] exists, it is easy to understand how important it would be to automatically identify the security patterns that are present in a software system. By knowing the security patterns present in a system and having a qualitative evaluation of each pattern, it would then be easy to make an estimate of the security features of the system under consideration.

In order to achieve the detection of the security patterns present in the system, the specification of these patterns using a specific notation is desirable as a first step. In this paper we make follow an approach to the specification the security patterns as a preprocessing step to their automatic detection in a software system. The remainder of the paper is organized as follows. Section 2 makes a short overview of existing security patterns while Section 3 describes our proposal for a first attempt to specify security patterns. Finally, in Section 4 we make some final conclusions and propose future directions for research.

Since the pioneer work by Yoder and Barcalow [20] several security patterns have been introduced in the literature. Though, there exists no clear definition of a security pattern because different authors refer to security patterns in a different context. For example, Ramachandran [16] refers to security patterns as basic elements of security system architecture in analogy to the work of Buschman et. al. [4] and Kis [12] has introduced security antipatterns. Romanosky [17], aims to investigate some security patterns using a specific format, in analogy to the examination of software design patterns [7]. Several authors describe security patterns intended for specific use, such as security patterns for web applications [19, 11], security patterns for agent systems [15], security patterns for cryptographic software [2], security patterns for mobile Java Code [14] and finally metadata, authentication and authorization patterns [5, 3]. Furthermore, the same security patterns appear in the literature with different names. Based on these facts, the Open Group Security Forum started a coordinated effort to build a comprehensive list of existing security patterns with the intended use of each pattern, all the names with which each security pattern exists in the literature, the motivation behind designing the pattern, the applicability of the pattern, the structure of the pattern, the classes that comprise the pattern, a collaboration diagram describing the sequence of actions for the use of the pattern, guidelines for when to use the pattern, descriptions of possible implementations of the pattern, known uses of the pattern and finally, related patterns [1]. The notion of a security pattern in the related technical guide published by the Open Group in March 2004 is completely in analogy with the notion of Design Patterns as originally stated by Gamma et. al. [7]. Our work is based on this review by Blakley et. al. [1] since this is the most comprehensive guide currently reviewing existing security patterns. For the sake of clarity, we will include in this paper the names of the patterns together with their intended use. We will also include a class diagram of the patterns. Blakley et. al. [1] divide security patterns in two categories. The first category is Available system patterns, which facilitate construction of systems that provide predictable uninterrupted access to the services and resources they offer to users. The second category is Protected system patterns, which facilitate construction of systems that protect valuable resources against unauthorized use, disclosure or modification.

2. A SHORT REVIEW OF EXISTING SECURITY PATTERNS

Since the pioneer work by Yoder and Barcalow [20] several security patterns have been introduced in the literature. Though, there exists no clear definition of a security pattern because different authors refer to security patterns in a different context. For example, Ramachandran [16] refers to security patterns as basic elements of security system architecture in analogy to the work of Buschman et. al. [4] and Kis [12] has introduced security antipatterns. Romanosky [17], aims to investigate some security patterns using a specific format, in analogy to the examination of software design patterns [7]. Several authors describe security patterns intended for specific use, such as security patterns for web applications [19, 11], security patterns for agent systems [15], security patterns for cryptographic software [2], security patterns for mobile Java Code [14] and finally metadata, authentication and authorization patterns [5, 3]. Furthermore, the same security patterns appear in the literature with different names. Based on these facts, the Open Group Security Forum started a coordinated effort to build a comprehensive list of existing security patterns with the intended use of each pattern, all the names with which each security pattern exists in the literature, the motivation behind designing the pattern, the applicability of the pattern, the structure of the pattern, the classes that comprise the pattern, a collaboration diagram describing the sequence of actions for the use of the pattern, guidelines for when to use the pattern, descriptions of possible implementations of the pattern, known uses of the pattern and finally, related patterns [1]. The notion of a security pattern in the related technical guide published by the Open Group in March 2004 is completely in analogy with the notion of Design Patterns as originally stated by Gamma et. al. [7]. Our work is based on this review by Blakley et. al. [1] since this is the most comprehensive guide currently reviewing existing security patterns. For the sake of clarity, we will include in this paper the names of the patterns together with their intended use. We will also include a class diagram of the patterns. Blakley et. al. [1] divide security patterns in two categories. The first category is Available system patterns, which facilitate construction of systems that provide predictable uninterrupted access to the services and resources they offer to users. The second category is Protected system patterns, which facilitate construction of systems that protect valuable resources against unauthorized use, disclosure or modification.

2.1 AVAILABLE SYSTEM PATTERNS

The intent of the Checkpointed System pattern is to structure a system so that its state can be recovered and restored to a known valid state in case a component fails. A class diagram of the pattern is shown in Figure 1. The Checkpointed System Pattern offers protection from loss or corruption of state information in case a component fails.

The intent of the Standby pattern is to structure a system so that the service provided by one component can be resumed from a different component. A class diagram of the pattern is shown in Figure 2. The Standby pattern can be used in cases where failed components may not be recoverable but a similar or identical backup component is available.

The intent of the Comparator-Checked Fault Tolerant System pattern is to structure a system so that an independent failure of one component will be detected quickly and so that an independent single-component failure will not cause a system failure. A class diagram of the pattern is shown in Figure 3. The operation of this pattern is more effective compared to the Checkpointed System Pattern and the Standby pattern, since faults that have not caused a failure yet can be detected by the Comparator.

The intent of the Replicated System pattern is to structure a system that allows provision from multiple points of presence and recovery in the case of failure of one or more components or links. A class diagram of the pattern is shown in Figure 4. The Workload Management Proxy assigns requests to the Replica that is in operation and has the smallest workload at the time of the decision.

The intent of the Error Detection/Correction pattern is to add redundancy to data to facilitate later detection of and recovery of errors. A class diagram of the pattern is shown in Figure 5. Data that are written by the Client are added redundant information by the Error Control Proxy and then saved to the Redundant Media/Link. In this way, when the Client makes a request to read the same data it is easy to check their integrity, before the read operation is performed.

2.2 PROTECTED SYSTEM PATTERNS

The intent of the Protected System pattern is to structure a system so that all access by clients is mediated by a guard that enforces a security policy. A class diagram of the pattern is shown in Figure 6. The Guard controls access requests to resources according to a predefined policy.

The intent of the Policy pattern is to isolate policy enforcement to a discrete component of an information system and to ensure that policy enforcement activities are performed in the proper sequence. A class diagram of the pattern is shown in Figure 7. The Policy class enforces rules that are to be applied by the Guard, for possible authentication. In the case of successful authentication Security Context attributes are set. After that, the Security Context is read from the guard and the guard requests a policy decision according to the rules enforced by the current Security Context.

The intent of the Authenticator pattern [3] is to perform authentication of a requesting process, before deciding access to distributed objects. A class diagram of the pattern is shown in Figure 8. If the authentication process performed by the Authenticator is successful the Authenticator forwards a request for the creation of a Remote Object to the ObjectFactory.

The intent of the Subject Descriptor pattern is to provide access to security-relevant attributes of an entity on whose behalf operations are to be performed. A class diagram of the pattern is shown in Figure 9. The Subject Descriptor pattern is used in conjunction with other security patterns to control the conditions under which authorization is to be performed. In more depth, it is used to represent authorization subjects as sets of predicates or assertions on attribute and property values.

The intent of the Secure Communication Pattern is to ensure that mutual security policy objectives are met when there is a need for two parties to communicate in the presence of threats. A class diagram of the pattern is shown in Figure 10. The Secure Communication pattern protects the communication channel. This is achieved through the use of the Communication Protection Proxy that checks all messages appropriately, before they are released to the Communications Channel.

The intent of the Security Context pattern is to provide a container for security attributes and data relating to a particular execution context, process, operation or action. A class diagram of the pattern is shown in Figure 11. After a process becomes active, an instance of Security Context is created by a Communication Protection Proxy and populated with the necessary information about the process. This information can then be used for authentication of the user initiating the process.

The intent of the Security Association pattern is to define a structure which provides each participant in a Secure Communication with the information it will use to protect messages to be transmitted to the other party and with the information it will use to understand and verify the protection applied to messages received from the other party. A class diagram of the pattern is shown in Figure 12. The Security Association pattern enables an instance of Secure Communication to protect more than one message.

Finally, the intent of the Secure Proxy pattern is to define the relationship between the guards of two instances of Protected System, in the case when one instance is entirely contained within the other. Figure 13 shows a class diagram of the pattern. This pattern organizes two lines of defense by using two consecutive Guards. The two Guards may check both on all the rules enforced by Policy in order to achieve increased protection.

3. AN APPROACH TO THE SPECIFICATION SECURITY PATTERNS

Our approach can be summarized in two steps. In the first step we model information that is vital to the automated design pattern detection process as a set of matrices. In the second step we have to add to the description security related information such as whether a specific class serves a specific security purpose e.g. being a Guard for resources.

3.1 REPRESENTATION OF CLASS DIAGRAMS AS MATRICES

The representation of the information present in a class diagram, that is vital to the detection of a security pattern, as a set of matrices that show the relations between classes present in the diagram, seems to be intuitively appealing. This will be illustrated by showing an example for each matrix type used in our representation. The matrices we use in our approach are an Association matrix showing associations between classes, an Aggregation matrix, a Generalization matrix, showing inheritance information and a Creation matrix. As an example for the Association matrix we will examine this matrix for the Checkpointed System pattern.

The corresponding matrix is shown in Table 1, where A=Recovery Proxy, B=Recoverable Component, C=Memento, D=Stateful Component.

Association	A	B	C	D
A	0	1	0	0
B	0	0	1	0
C	0	0	0	0
D	0	0	0	0

Table 1. The Association matrix for the Checkpointed System pattern.

As an example of the Aggregation matrix we will examine this matrix for the Comparator-Checked Fault Tolerant System Pattern. The corresponding matrix is shown in Table 2, where A = Component, B = Recoverable Component 1, C = Recoverable Component 2, D = Memento 1, E= Memento 2, F = Comparator.

Aggregation	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	1	1	0

Table 2. The Aggregation matrix for the Comparator-Checked Fault Tolerant System Pattern

The Aggregation matrix has a 1 in the row for the Comparator and the columns of the Mementos since the Comparator may compare aggregations of Mementos. As an example of the Generalization matrix we will examine this matrix for the Checkpointed System Pattern. The corresponding matrix is shown in Table 3, where A = Recovery Proxy, B = Recoverable Component, C = Memento, D = Stateful Component.

Generalization	A	B	C	D
A	0	0	0	0
B	0	0	0	1
C	0	0	0	0
D	0	0	0	0

Table 3. The Generalization matrix for the Checkpointed System pattern.

The Generalization matrix has a 1 in the row for the Recoverable Component and the column of the Stateful Component since the Stateful Component is a generalization of the Recoverable Component. As an example of the Generalization matrix we will examine this matrix for the Authenticator Pattern. The corresponding matrix is shown in Table 4, where A = Concrete Authenticator, B = Authenticator, C = Concrete ObjectFactory, D = ObjectFactory, E = Remote Object.

Creation	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	1
D	0	0	0	0	0
E	0	0	0	0	0

Table 4. The Creation Matrix for the Authenticator pattern.

The Creation matrix has a 1 in the row for the Concrete ObjectFactory and the column of the Remote Object since the Concrete ObjectFactory creates Remote Objects.

3.2 Addition of Security Related Information and Examination of a Sketch for a Security Pattern Detection Algorithm

The representation of class diagrams as matrices is the first step in the specification of the Security Patterns. This representation offers us only structural information about the class diagram. The Security Patterns though, possess information in the form of attributes that is vital to their representation. Specifically, the main operation of classes can be identified through the class labels. So, in order to achieve the detection of these patterns in a software system, additional information related to the security characteristics of these patterns should be incorporated. As an example, in the Comparator Checked Fault Tolerant System pattern we should minimally incorporate into the specification which of the classes are Mementos and which class is the Comparator. The obvious way to achieve security pattern detection would be to propose a two phase algorithm. In the first phase the matching of structural information would be examined, through the comparison of the set of matrices for the system under consideration with the set of matrices for the security pattern to be detected. If a possible match would be found, then, in a second phase the attribute information of the classes would have to be compared, in order to decide whether an exact match has occurred.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have examined the importance of Security Patterns to a software security system. Furthermore, we proposed a method to specify Security Patterns, using a sets of matrices representation and attribute information of the classes. A sketch of an algorithm to detect security patterns has been presented. Future work includes the definition of the algorithm itself and its evaluation in a real software system.

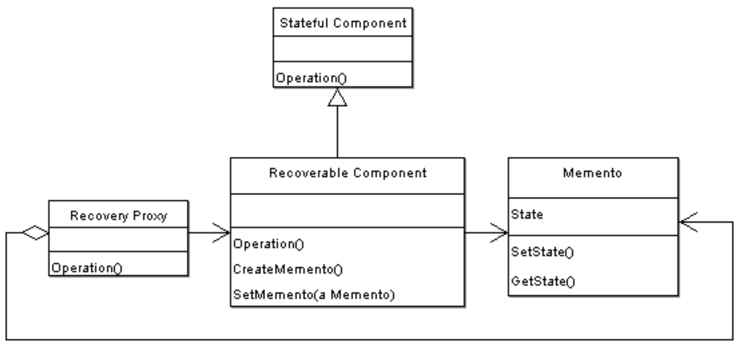

Fig. 1. Class Diagram of the Checkpointed System Pattern.

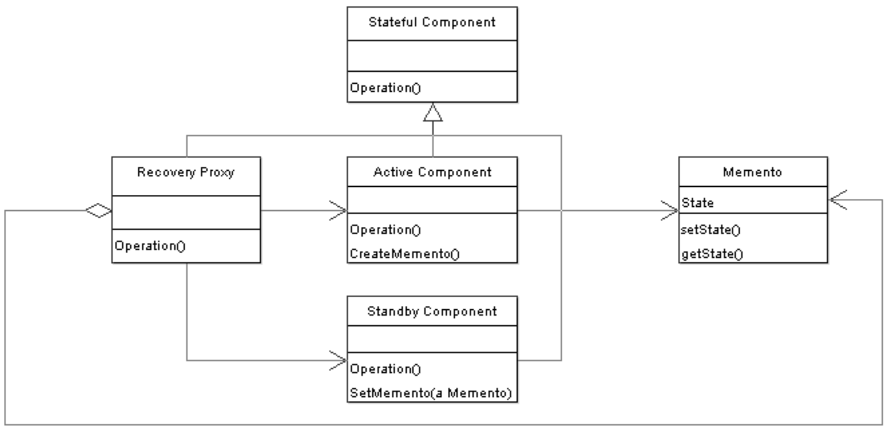

Fig. 2. Class diagram of the Standby pattern

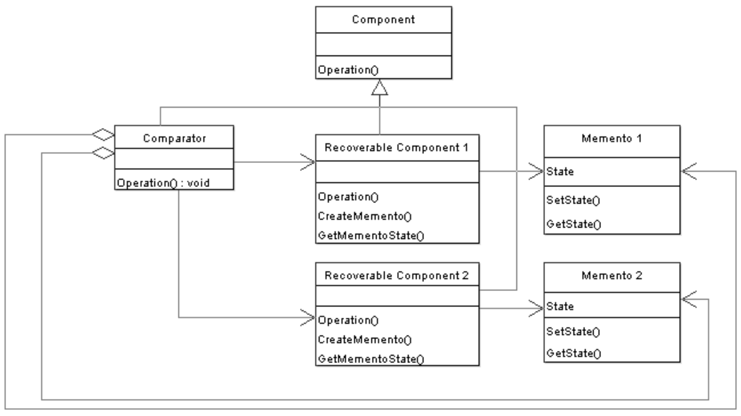

Fig. 3. Class diagram of the Comparator-Checked Fault-Tolerant System Pattern

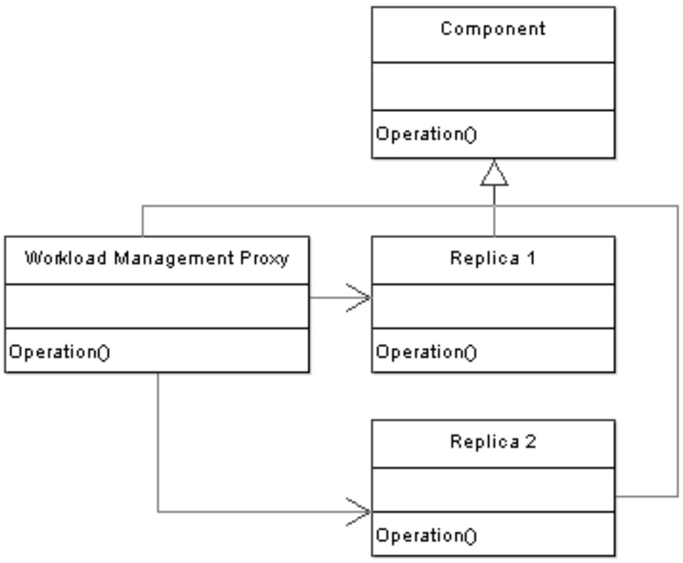

Fig. 4. Class diagram of the Replicated System pattern

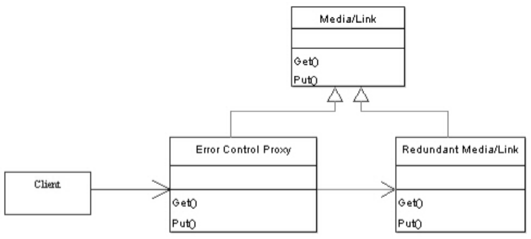

Fig. 5. Class diagram of the Error Detection/Correction pattern

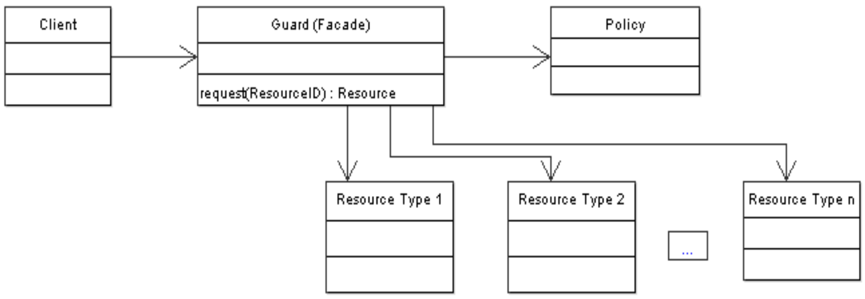

Fig. 6. Class diagram of the Protected System pattern

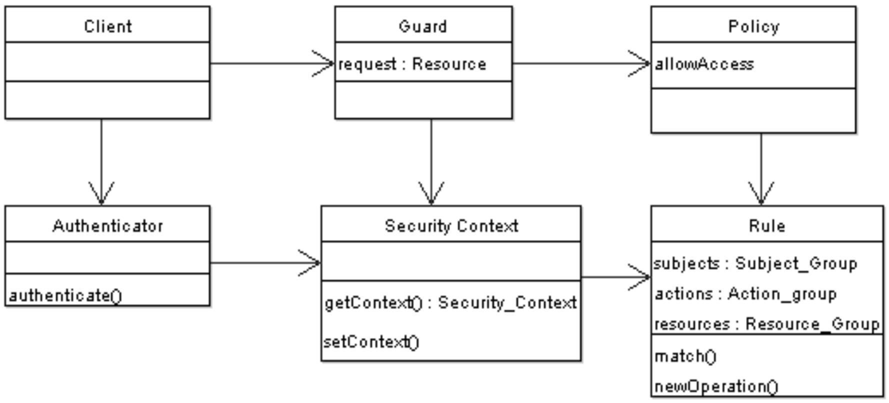

Fig. 7. Class diagram of the Policy pattern

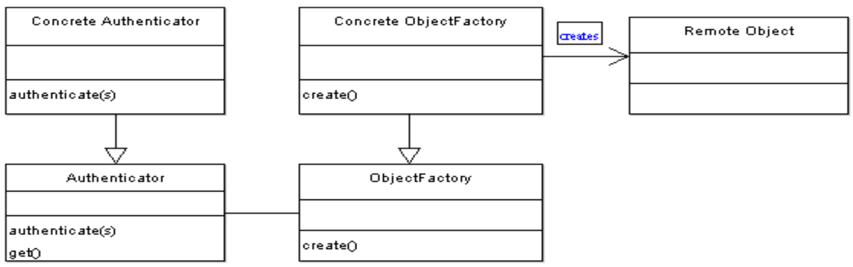

Fig. 8. Class diagram of the Authenticator pattern

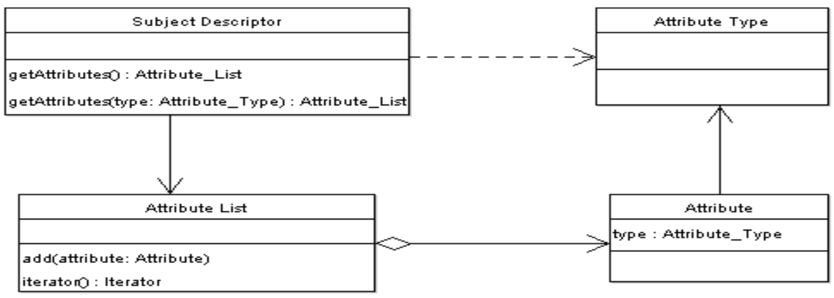
Subject Descriptor
getAttributes() : Attribute_List
getAttributes(type: Attribute_Type) : Attribute_List
Attribute Type
Attribute List
add(attribute: Attribute)
iterator() : Iterator
Attribute
type : Attribute_Type

Fig. 9. Class diagram of the Subject Descriptor Pattern

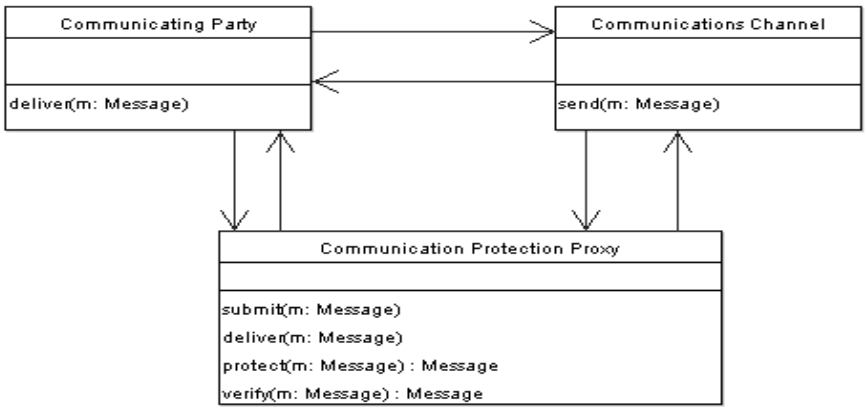

Fig. 10. Class diagram of the Secure Communication Pattern

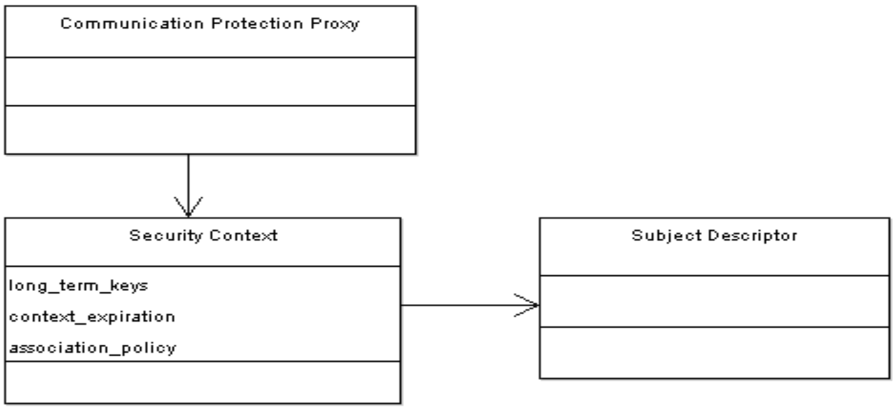

Fig. 11. Class diagram of the Security Context pattern

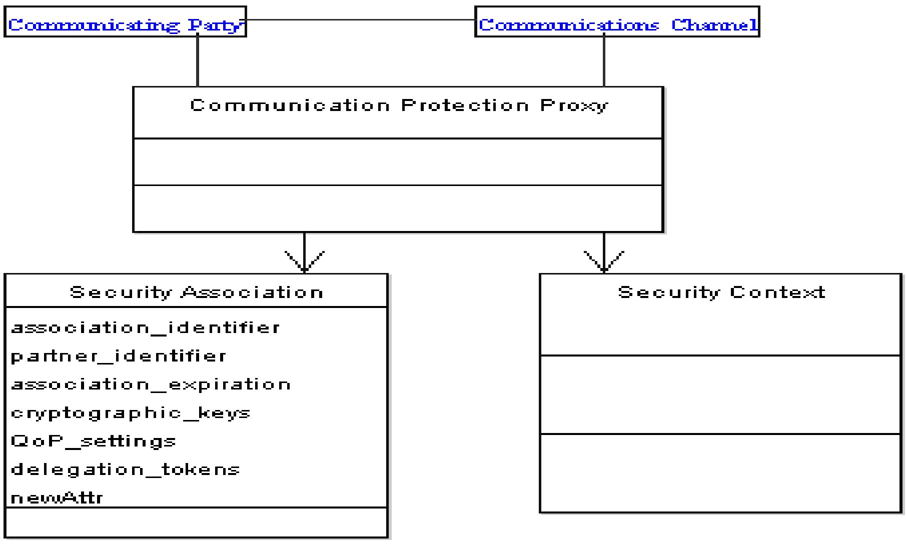

Fig. 12. Class diagram of the Security Association Pattern

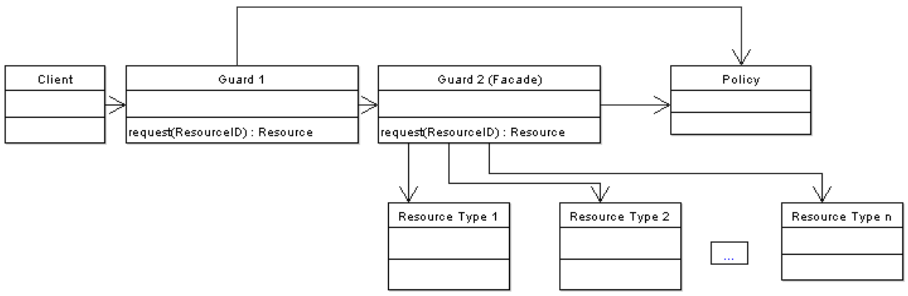

Fig. 13. Class diagram of the Secure Proxy pattern

References

[1] **B. Blakley, C. Heath and Members of the Open Group Security Forum**, Security Design Patterns (Open Group Technical Guide, 2004)

[2] **A. Braga, C. Rubira, R. Dahab R.**, Tropyc: A Pattern Language for Cryptographic Software, Proceedings of the 5th Conference on Pattern

Languages of Programming (PLoP '98), 1998

[3] **F. Lee Brown, J. Di Vietri, G. Diaz de Villegas, E. Fernandez** , The Authenticator Pattern, Proceedings of the 6th Conference on Pattern Languages of Programming (PLoP '99), 1999

[4] **F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, M Stahl**, Pattern Oriented Software Architecture - A System of Patterns, (John Wiley and Sons, 1996)

[5] **E. Fernandez** , Metadata and authorization patterns, (http://www.cse.fau.edu/ ed/ Metadata Patterns.pdf, 2000)

[6] **P. Fites, M. Kratz** , Information Systems Security: A Practitioner's Reference, (International Thomson Computer Press, 1996)

[7] **E. Gamma, R. Helm, R. Johnson, J. Vlissides** , Design Patterns (Addison Wesley, 1995)

[8] **S. T. Halkidis, A. Chatzigeorgiou, G Stephanides**, A Qualitative Evaluation of Security Patterns, in Proceedings of the Sixth International Conference on Information and Communications Security, (ICICS '04), (Malaga, 27-29 October 2004, Lecture Notes in Computer Science 3269)

[9] **M. Howard, D LeBlanc**, Writing Secure Code, (Microsoft Press, 2002)

[10] **IBM**, Introduction to Business Security Patterns, (IBM White Paper 2003)

[11] **D. Kienzle, M. Elder**, Security Patterns for Web Application Development, (Univ. of Virginia, Technical Report, 2002)

[12] **M. Kis**, Information Security Antipatterns in Software Requirements Engineering, In Proceedings of the 9th Conference on Pattern Languages of Programming (PLoP '02),2002

[13] **M. Krause, H. Tipton** , (Eds.), Information Security Management Handbook, Fourth Edition, (CRC Press - Auerbach Publications, 1999)

[14] **Q. Mahmoud** , Security Policy: A Design Pattern for Mobile Java Code, in Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP '00), 2000

[15] **H. Mouratidis, P. Giorgini, M. Schumacher**, Security Patterns for Agent Systems, in Proceedings of the Eighth European Conference on Pattern Languages of Programs (EuroPLoP '03), 2003

[16] **J. Ramachandran**, ., Designing Security Architecture Solutions, (John Wiley and Sons, 2002)

[17] **S. Romanosky**, Security Design Patterns (http://www.romanosky.net/papers/ securityDesign/Patterns.html, 2002)

[18] **J. Viega, G. McGraw**, Building Secure Software, How to Avoid Security Problems the Right Way (Addison Wesley, 2002)

[19] **M. Weiss**, Patterns for Web Applications, in Proceedings of the 10th Conference on Pattern Languages of Programming (PLoP '03), 2003

[20] **J. Yoder, J. Barcalow**, ., Architectural Patterns for enabling application security, in Proceedings of the 4th Conference on Pattern Languages of Programming (PLoP '97), 1997