Comparison of Security Patterns

David G. Rosado, , Eduardo Fernández-Medina, Mario Piattini

ALARCOS Research Group. Information Systems and Technologies Department
UCLM-Soluziona Research and Development Institute. University of Castilla-La Mancha
Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain

Carlos Gutierrez

STL. Calle Manuel Tovar 9, 28034 Madrid, Spain

Summary
Security patterns are a recent development as a way to encapsulate the accumulated knowledge about secure systems design, and security patterns are also intended to be used and understood by developers who are not security professionals. In this paper, we will compare several security patterns to be used when dealing with application security, following an approach that we consider important for measuring the security degree of the patterns, and indicating a fulfilment or not of the properties and attributes common to all security systems. We will see that these patterns present some weaknesses. Although they fulfil the design original intention, they don't fulfil many security attributes.
Key words:
Security, Security Patterns, Security Architecture

Introduction

It is very common not to consider security in the first stages of systems development but to deal with it once the system has been designed and implemented. However, those aspects known as "quality requirements" [5, 8], being security one of them, must be described in a concrete way before the system architecture is designed [4]. Ignoring security issues is dangerous because it can be difficult to retrofit security in an application [15].

Security patterns are proposed as a means of bridging the gap between developers and security experts. Security patterns are intended to capture security expertise in the form of worked solutions to recurring problems. Security patterns are also intended to be used and understood by developers who are not security professionals [12]. The first person who used the pattern approach was Christopher Alexander [2], and in his book he indicated that each pattern describes a problem which occurs over and over again in our environment, and then states the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. The "Gang of Four" book, as it is commonly known, defined design patterns as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context" [10].

This paper will study a series of security patterns that help us implement security requirements in the applications design. They are patterns that guide the systems design to make them more secure in a comfortable and efficient way. The rest of the paper is organized as follows. In section 2, we will define a template to define patterns; in section 3, we will describe each one of the selected patterns in order to make the comparison study. Then, we will describe the comparison framework that we have used and we will perform the patterns comparison. Finally, we will put forward our conclusions.

2. Security Pattern Template

It is advisable that a software pattern is organized into multiple sections (the total set will be known as template) that deal with different aspects such as name, problem and solution. Templates can be defined as we like but always maintaining the main categories. Thus, each author can describe all sections he/she considers important according to his/her viewpoint [14].

In this section, it will be defined a template formed by the following sections (based on [1, 10]) i) *Intent*: It describes what the pattern does, which its rationale and intent are, and what particular design issue it addresses. ii) *Context*: It describes the context of the problem. iii) *Problem*: It gives a statement of the problem that this pattern solves. iv) *Description*: A scenario that illustrates a design problem. v) Solution: To give a statement of the solution to the problem. vi) *Consequences*: To describe the trade-offs and results when we use the pattern. vii) *Known uses*: Examples of the patterns found in real systems viii) *Related patterns*: To list other related patterns that use this pattern as a reference.

3. Security Patterns Selected

Once the template has been defined, we select a set of patterns to perform a study of their characteristics and find out the degree of security that they supply to the systems that use them. These patterns are as follows: 1) Authorization Pattern [7]; 2) RBAC Pattern (Role-Based Access Control) [7]; 3) Multilevel Security Pattern [7]; 4) File Authorization Pattern [6]; 5) Virtual Address Space Access Control [6]; 6) Reference Monitor Pattern [6]; 7) SAP Pattern (Single Access Point), [14, 15]; 8) Check Point Pattern [14, 15]; y 9) Session Pattern [14, 15].

Due to space constraints, we will not consider all the sections of the template but only those sections that we consider relevant to clearly define the considered pattern.

3.1 Authorization Pattern

i) *Intent*: It describes who is authorized to access the resources systems ii) *Context*: Any environment where we need to control the access to computing resources. iii) *Problem*: The permissions granted for security subjects that have access to protected objects need to be explicitly indicated. On the contrary, any subject could access any resource. iv) *Description*: To structure the different access policies, we distinguish between active entities (subjects) and passive resources (protection objects). v) *Solution*: The Authorization structure (see Fig. 1) can be captured from classes and relationships or associations. vi) *Consequences*: The solution is independent of the resources to be protected. The subjects can be executions of processes, users, roles and group of users; the objects to be protected can be transactions, memory area, I/0 devices, files or other resources of the operating system and the type of access can be reading, writing, execution or methods in higher level objects.

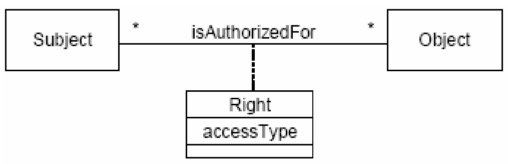

Fig 1. Authorization Pattern. The active entities are represented by the Subject class and the passive resources (or resources to be protected) are represented the by Object class. The relationship between subject and object describes what subject is authorized to access certain objects (Rights).

3.2 RBAC Pattern

i) *Intent*: To control the access resources only based on the subject role. ii) *Context*: Any environment where we need to control the access to computing resources and where users can be classified according to their jobs and tasks. iii) *Problem*: It is necessary to assign rights and permissions (central authority) in an appropriate way for users to be able to access the protected objects. iv) *Description*: It improves the administration by using roles that can be assigned to individual users or groups. v) *Solution*: It extends the idea of the *Authorization* pattern by translating roles as subjects. A basic model for RBAC is shown in Fig. 2. Users are assigned to roles, roles are given rights according to their functions and the *Right* association class defines the types of access that a user within a role is authorized to apply to the protection object. vi) *Consequences*: When introducing roles, the administrative effort is reduced because there is no need of assigning rights to individuals. The roles structure let us manage big groups as well as reduce rules.

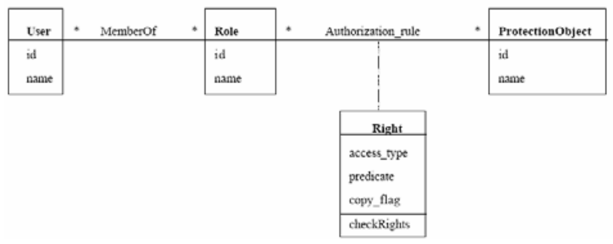

Fig. 2. RBAC Pattern.User and Role classes describe registered users and predefined roles, respectively. The combination Role, Protection Object and Rights is an instance of the Authorization pattern.

3.3 Multilevel Security Pattern

i) *Intent*: It provides a mechanism of access management in a system with several levels of security classification. ii) *Context*: It is applicable to systems that need to provide several security levels. iii) *Problem*: How to decide access in an environment with security classifications. iv) *Description*: In many systems, data integrity and confidentiality need to be guaranteed. This model would be able to be used in any architecture level and it provides a structure that allows us to have differente security levels for both subjects and objects. v) *Solution*: To represent the structure of Multilevel Security, there must be an instance of the class Subject Clasification for each subject and an instance of the class Object Classification for each object (see Fig. 3). These instances are used to add levels and objects security categories to a subject. vi) *Consequences*: It facilitates the administrative work in an environment that requires the classification of subjects and objects. The multilevel security can be expensive since subjects and objects need to be classified into certain levels of sensitiveness.

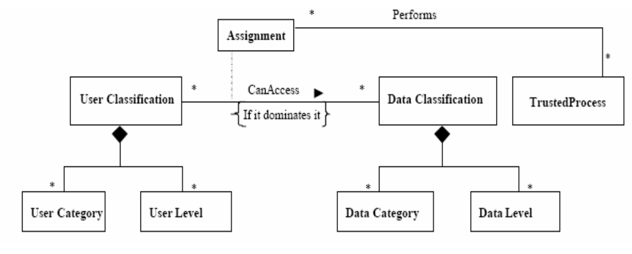

Fig. 3. Multilevel Security Pattern

3.4 File Authorization Pattern

i) *Intent*: To control the access to files in the operating system. ii) *Context*: Operating systems users need to use files to store information and access must be restricted only to authorized users. iii) *Problem*: Files can contain important information and the access to them must be carefully controlled. As files can be shared, it is difficult to impose security. iv) *Description*: There can be different categories of subjects (users, roles and groups). All these subjects can be uniformly managed and must be authorized to access files, directories and workstations. v) *Solution*: To specialize the Authorization pattern as it is shown in Fig. 4. vi) *Consequences*: It can contain a variety of subjects (users, roles and groups) that can be structured in a recurrent way. Access objects can be simple files or directories or recurrent structures of directories and files; and all files within a directory can have the same types of access.

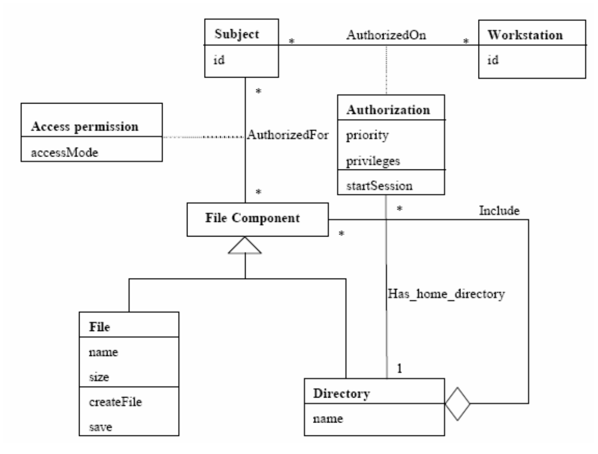

Fig. 4. File Authorization Pattern. Two versions of the Authorization pattern 1) Replacing objects by files or directories, and rights by access permissions and 2) the same subject and objects are replaced by workstations.

3.5 Virtual Address Space Access Control Pattern

i) *Intent*: To control the access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access types. ii) *Context*: Each process is executed in its own address space. The allowed accesses are reading, writing and executing, and other types. iii) *Problem*: Processes must be controlled when they access memory, otherwise they could overwrite areas from other processes or gain access to private information. iv) *Description*: There is a variety of structures of virtual memory addresses space: some systems use a separate set, others an only level address space. Furthermore, VAS can be divided into users and operating system. We would like to control the access to all these kinds in a uniform way. This implies that an implementation of the solution will require a specific hardware architecture. However, the solution must be independent of the hardware. v) *Solution*: To divide VAS into segments corresponding to logical units within the programs. To use descriptors to indicate access rights such as the beginning address of the accesible segment, the limit of the accesible segment and the type of allowed access (reading, writing, executing). Fig. 5 shows a diagram to indicate the solution to the class. vi) *Consequences*: This pattern provides a protection of the required segment because a process cannot access a segment without an own descriptor. If all resources are outlined in a virtual address space, the pattern can control the access to any kind of resource, including files. The solution is dependent of the hardware.

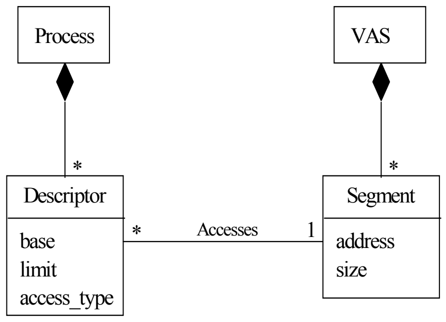

Fig. 5. Virtual Address Space Access Control Pattern. A process (Process class) must have a descriptor (Descriptor class) to access a segment in the VAS.

3.6 Reference Monitor Pattern

i) *Intent*: To make it possible that all authorizations are fulfilled when a process requires resources. ii) *Context*: A multiprocess environment making petitions by resources. iii) *Problem*: If the defined authorizations are not fulfilled, processes can execute all kind of illegal actions, for instance, any user could read any file. iv) *Description*: To define authorization rules is not enough; these rules must be imposed when a process makes a petition to a resource. There are many implementations and we need an abstract execution model. v) *Solution*: To define an abstract process that intercepts all petitions from resources and confirms them. Fig. 6 shows us a class diagram in which we can see a Reference Monitor. vi) *Consequences*: If all petitions are intercepted, we can assure that they fulfil the rules. The specific implementations are necessary for any kind of resource. To check each petition can mean a performance loose.

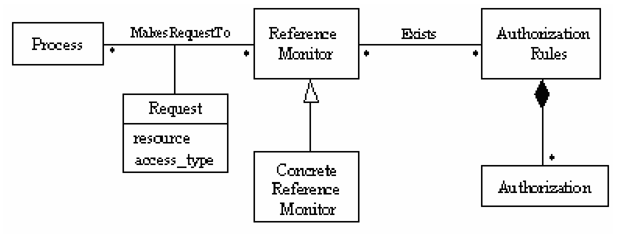

Fig. 6. Reference Monitor Pattern. Authorization rules indicate a collection of authorization rules organized as ACLs (access control lists)

3.7 Session Pattern

i) *Intent*: To provide us with an environment where a user's rights can be restricted and controlled. ii) *Context*: Any environment where we need to control the access to computing resources. iii) *Problem*: Depending on the context, for example, within a certain application, a user will only activate a subset of the authorizations he/she has. This will avoid that users use their rights wrongly (for instance, to accidentally delete certain files). In this way, if an attacker endangers a process, the damage potential is reduced. iv) *Description*: In many systems, global information is necessary in several points. To overcome this problem, Session objects that provide the necessary information are used. v) *Solution*: Fig. 7 shows us elements of a class diagram session. A subject can be in several sessions at the same time and it has a limited lifetime. When we start a session (for example, when registering ourselves), a user only activates a set of authorization contexts assigned to him/her, then, only the necessary rights are available within this session. vi) *Consequences*: Each session gains all privileges that are necessary to carry out the desired tasks. Thus, damage will be potentially reduced when a session is in danger because only an activated subset of authorization can be wrongly used.

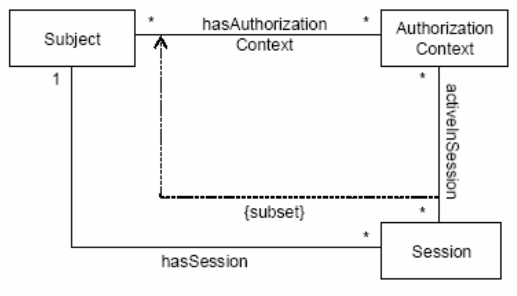

Fig. 7. Session Pattern. The Subject class describes an active entity that accesses the system and asks for resources. The AuthorizationContext class describes a set of contexts of executions or active rights that the user has in a given interaction.

3.8 SAP Pattern

i) *Intent*: The Single Access Point pattern defines one single interface for all communication with system external entities in order to improve control and monitoring. ii) *Context*: SAP can be applied to self-contained systems that need to communicate with external entities. It can be used at the application-level as well as at the host or network-levels, even though implementation at the abstraction-level of application development is more apparent at first glance. Application at the network-level implies that all sub-nets inside the system's boundaries are isolated from other nets. The only connection to the outside is a Single Access Point. While we assume virtual isolation of system internal entities in high abstraction levels, lower design levels have to include this goal in their models (for example by adding encryption, signing of messages, and tokens that guarantee freshness). The system's deployment structure determines where further securing effort is necessary. iii) *Problem*: A security model is difficult to confirm when it has multiple main, back and lateral doors to come in the application. iv) *Description*: Due to various access points, many systems cannot be protected effectively against attacks from the outside. Hidden back doors and different (inconsistent) implementations of security policies aggravate protection. The application of the Single Access Point pattern prevents external entities from communicating directly with components in the system. All inbound traffic is routed through one channel, where monitoring can be performed easily. Additionally, the Single Access Point is an appropriate place for capturing an information log on the parties currently accessing the system. This data may be useful inside the system to verify certain access requests and to determine their rights. v) *Solution*: SAP represents the only connection of the system with outside (see Fig. 8). All incoming communication petitions are taken to the SAP instance that works as a mediator. If certain policies need to be imposed, all petitions should be sent to a *Check Point* class before they are transmitted to their addresses. vi) *Consequences*: SAP will provide a good place to capture register information as well as to carry out authorization tasks. The undesirable modification of data can be avoided with efficient checks that let us access the system. Confidentiality can also be improved as disclosure of information to unauthorized parties is more unlikely. Every access will pass the Single Access Point and can be monitored. Undesirable modification of data can be prevented better by efficient checks who is allowed to access the system (integrity). Availability may be reduced if the Single Access Point cannot handle all accesses concurrently. Denial of Service (DoS) Attacks can be prevented more efficiently. All information that is necessary for their detection can be gathered at the Single Access Point.

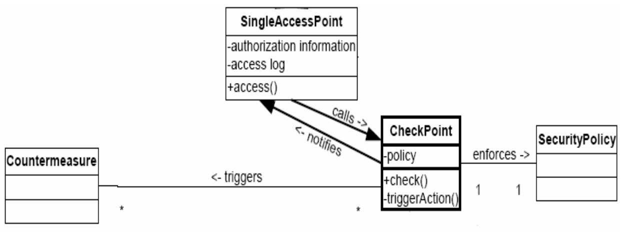

Fig. 8. SAP Pattern and Check Point Pattern

3.9 Check Point Pattern

i) *Intent*: It states a structure to check the incoming petitions. If it finds violations, this pattern is in charge of taking the appropriate countermeasures. ii) *Context*: *Check Points* are applicable in any security-relevant communication. It can be used at each abstraction-level from inside-application-level to network-level. In order to perform a check, the system needs to have a policy that will be enforced. The Check Point implementing that policy should be able to distinguish between user mistakes and malicious attacks. iii) *Problem*: In order to prevent unauthorized access it is vital to check who interacts in which manner with a system. It can be a difficult task to determine whether a given access should be granted or not. Any secure system needs a component that monitors the current communication and takes measures if necessary. iv) *Description*: It needs to take any kind of action, if there are mistakes depending on the seriousness. v) *Solution*: A *Check Point* is a component that analyzes all petitions and messages. A SAP is predestined to be combined with a Check Point for all messages to be supervised (see Fig. 8). It implements a method to check messages according to the current security policy. It gives place to actions that could be necessary to protect the system against attacks. vi) *Consequences*: Its application can benefit the system confidentiality, if the checking algorithm is correct. Undesirable modifications can be filtered if the checking algorithm is able to detect those attacks. Complex checking routines can make both the system and the message interchange work slower. *Denial of Service* (DoS) attacks can be prevented, if the *Check Point* algorithm takes appropriate actions. Maintenance of security-relevant code will be easier if it is located at one position. Though, complexity of the check algorithm is, depending on the implemented policy, high. Some communication activity might be prevented even if it is not harmful. A high-quality check algorithm is vital.

4. Comparative Framework

In this section, we will put forward a comparison based on certain criteria that we consider important for security with the purpose of distinguishing all properties and characteristics of all previous patterns as well as showing a general vision of the subject. There are some comparisons [14] of patterns with certain criteria or security principles [13]. Some of these defined criteria are based on the works of Babar [3] and Firesmith [9], in which they select the most commonly used attributes and security properties in the security dominion. The used criteria to make our comparison are the following: *Authentication*: It must be validated the identity of customers to frustrate any disauthorized access. *Authorization*: This attribute defines the access privileges of entities to different resources and services of a system. *Integrity*: To guarantee that data and communications will not be compromised by active attacks. *Confidentiality*: The guarantee that information is not accessed by disathorized parts. *Attacker detection*: To be able to detect and register access or modification intents in the system coming from disauthorized users. *No-Repudiation*: It prevents that certain participant in certain interaction can deny to have participated in it. *Auditability*: To let the security staff audit the state and use of the security mechanisms. *Maintainability*: It facilitates the introduction or modification of the security policy during the software development life cycle. *Availability*: It assures that authorized users can use the resources when they are required. *Reliability*: It assures the system operations due to failures or configuration mistakes. Besides, it assures the system availability even when the system is being attacked. *Error management*: A system must provide a robust error management mechanism. *Performance*: It indicates the impact of the pattern on the functioning of a system. *Implementation cost*: Costs accompanying the pattern use. *Security degree*: It indicates the security level that the pattern has for the function it fulfils.

Table 1. Patterns Comparative Table for the selected security criteria

H : High;
M : Medium;
L : Low;
? Reduce;
? : Increase;

	Authentication	Authorization	Integrity	Confidentiality	Attackers detection	No-Repudiation	Auditability	Maintainability	Availability	Reliability	Error management	Performance	Implem. Cost	Security degree
Authorization		χ		χ	χ^1			χ	χ			—[a]	L	M
RBAC		χ		χ	χ^1			χ	χ			†[c]	M	M
Multilevel		χ	χ^5	χ^6	χ^1				χ	χ		—[d]	H	H
File Access		χ		χ	χ^1			χ	χ			†[e]	L	M
Virtual Address		χ	χ^7	χ^7	χ^1				χ			†[f]	L	M
Reference Monitor		χ	χ^7	χ^7	χ^1				χ	χ^7		—	H	H
SAP	χ	χ	χ^2	χ	χ	χ	χ		χ	χ	χ	—[b]	H	H
Check Point	χ	χ	χ^2	χ^2	χ	χ			χ	χ	χ	—[b]	H	H
Session		χ	χ	χ	χ^1			χ^3	χ	χ^4		†[c]	H	H

1. Only detection. 2. efficient check algorithm. 3. first step development. 4. Subset of the authorizations activated. 5. Bilba model. 6. Bell LaPadula model. 7. To process level. a) Many users. b) complex checks. c) Efficient implementation. d) Evaluation access rights. e) Tree of directories. f) If it uses Reference Monitor.

As it can be seen in Table 1, many patterns share the same security properties but each one of them is designed for a specific function. There are not two different patterns having the same purpose; some patterns use others as a basis, amplifying the design and incorporating into this new pattern a new characteristic that improves security. The majority of found patterns are based on guaranteeing access control, supplying confidentiality and in some cases, also integrity and reliability, but they do not take into account properties as important as error management, flexibility or maintenance, etc. There are patterns with a high degree of security but they are complex patterns. Then, if we want to have a system with a high degree of security, they will be also more complex systems, affecting their performance. Developers (not security experts) can find many security patterns but it is very difficult to determine which pattern is better to be used or which pattern guarantees certain degree of security. For this reason, we find a lack of a method or a flexible model of security architectures that guarantees security of the system in many aspects and that guides developers in the right way for the security implementation in their systems, according to the specific requirements of them [11].

Our future work will be that of studying the different security architectures existing in the systems design together with defining a method to specify flexible security architectures that can be easily adapted to systems with very different security requirements as well as guarantee security.

Acknowledgements

This research is part of the following projects: DIMENSIONS (PBC-05-012-2) financed by FEDER and by the “Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha” (Spain), RETISTIC (TIC2002-12487-E) and CALIPO (TIC2003-07804-C05-03) granted by the “Dirección General de Investigación del Ministerio de Ciencia y Tecnología” (Spain).

References

[1] AGCS, "AG Communication System. Template Pattern," 1996.

[2] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language: towns, builings, construction*. New York: Oxford University Press, 1977.

[3] M. A. Babar, X. Wang, and I. Gorton, "Supporting Security Sensitive Architecture Design," presented at QoSA-SOQUA 2005, 2005.

[4] M. R. Barbacci, R. Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, et al., "Quality Attribute Workshops (QWAs). Third Edition.," Carnegie Mellon. Software Engineering Institute. CMU/SEI-2003-TR-016, August 2003 2003.

[5] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice," 2nd ed: Addison-Wesley, 2003

[6] E. B. Fernandez, "Patterns for Operating Systems Access Control," presented at 9th Conference on Pattern Languages of Programs, PLoP 2002, Allerton Park, Illinois, USA, 2002.

[7] E. B. Fernandez and R. Pan, "A pattern language for security models," presented at 8th Conference on Pattern Languages of Programs, PLoP 2001, Allerton Park, Illinois, USA, 2001.

[8] D. G. Firesmith, "Commom Concepts Underlying Safety, Security, and Survivability Engineering," SEI CMU/SEI-2003-TN-033, December 2003 2003.

[9] D. G. Firesmith, "Specifying Reusable Security Requirements," *Journal of Object Technology*, vol. 3, pp. 61-75, 2004.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley, 1994.

[11] C. Gutiérrez, E. Fernández-Medina, and M. Piattini, "Towards a Process for Web Services Security," presented at WOSIS'05, Miami, Florida, USA, 2005.

[12] D. M. Kienzle and M. C. Elder, "Final Technical Report: Security Patterns for web Application Development," February 2005.

[13] J. Viega and G. McGraw, *Building Secure Software - How to Avoid Security Problems the Right Way.*, 1st ed: Addison-Wesley, 2002.

[14] R. Wassermann, "Using Security Patterns to Model and Analyze Security Requirements," 032.04/E, 9th March 2004.

[15] J. Yoder and J. Barcalow, "Architectural Patterns for Enabling Application Security," presented at 4th Conference on Patterns Language of Programming, PLop 1997, Monticello, Illinois, USA, 1997.

David G. Rosado has an MSc in Computer Science from the University of Málaga (Spain) and currently he is a PhD Student at the University of Castilla-La Mancha. His research activities are focused on security architectures for Information Systems. He has published several papers in national and international conferences on these subjects. He is a member of the ALARCOS research group of the Information Systems and Technologies Department at the University of Castilla-La Mancha, in Ciudad Real, Spain. His e-mail address is: david.grosado@uclm.es.

Carlos Gutiérrez has an MSc from the Autonomous University of Madrid (Spain) and currently he is a PhD candidate and Assistant Professor at the University of Castilla-La Mancha. He has developed his professional activities in national and international companies doing consultancy work. He is currently an Internet analyst in Sistemas Técnicos de Loterías del Estado (State Lotteries' Technical Systems). His research activities are focused on web services security and secure software architectures. He has published several papers in international conferences and diverse articles in national and international journals on these subjects. He is participating at the ALARCOS research group and he is an ACM member. His e-mail address is: carlos.gutierrez@stl.es.

Eduardo Fernández-Medina holds a PhD. and an MSc. in Computer Science from the University of Sevilla. He is Assistant Professor at the Escuela Superior de Informática of the University of Castilla-La Mancha at Ciudad Real (Spain), his research activity being in the field of security in databases, datawarehouses, web services and information systems, and also in security metrics. Fernández-Medina is co-editor of several books and chapter books on these subjects, and has several dozens of papers in national and international conferences (DEXA, CAISE, UML, ER, etc.). Author of several manuscripts in national and international journals (Information Software Technology, Computers And Security, Information Systems Security, etc.), he is a member of the ALARCOS research group of the Information Systems and Technologies Department at the University of Castilla-La Mancha, in Ciudad Real, Spain. He belongs to various professional and research associations (ATI, AEC, ISO, IFIP WG11.3 etc.). Eduardo's e-mail is eduardo.fdezmedina@uclm.es.

Mario Piattini has an MSc and a PhD in Computer Science from the Politechnical University of Madrid. He is a Certified Information System Auditor from the ISACA (Information System Audit and Control Association). Full Professor at the Escuela Superior de Informática of the Castilla-La Mancha University (Spain) and author of several books and papers on databases, software engineering and information systems, Piattini leads the ALARCOS research group of the Information Systems and Technologies Department at the University of Castilla-La Mancha, in Ciudad Real, Spain. His research interests are: advanced database design, database quality, software metrics, object- oriented metrics and software maintenance. His e-mail address is Mario.Piattini@uclm.es.