Lecture Notes in Computer Science

The LNCS series reports state-of-the-art results in computer science research, development, and education, at a high level and in both printed and electronic form. Enjoying tight cooperation with the R&D community, with numerous individuals, as well as with prestigious organizations and societies, LNCS has grown into the most comprehensive computer science research forum available.

The scope of LNCS, including its subseries LNAI and LNBI, spans the whole range of computer science and information technology including interdisciplinary topics in a variety of application fields. The type of material published traditionally includes

- proceedings (published in time for the respective conference)
- post-proceedings (consisting of thoroughly revised final full papers)
- research monographs (which may be based on outstanding PhD work, research projects, technical reports, etc.)

More recently, several color-cover sublines have been added featuring, beyond a collection of papers, various added-value components; these sublines include

- tutorials (textbook-like monographs or collections of lectures given at advanced courses)
- state-of-the-art surveys (offering complete and mediated coverage of a topic)
- hot topics (introducing emergent topics to the broader community)

In parallel to the printed book, each new volume is published electronically in LNCS Online.

Detailed information on LNCS can be found at
www.springer.com/lncs

Proposals for publication should be sent to
LNCS Editorial, Tiergartenstr. 17, 69121 Heidelberg, Germany
E-mail: lncs@springer.com

ISSN 0302-9743
ISBN 978-3-540-74482-5
9 783540 744825

Lecture Notes in Computer Science
LNCS LNAI LNBI

springer.com

Gervasi • Gavrilova (Eds.)

LNCS 4707

Computational Science and Its Applications – ICCSA 2007

3 Part III

ICCSA

LNCS 4707

Osvaldo Gervasi
Marina L. Gavrilova (Eds.)

Computational Science and Its Applications – ICCSA 2007

International Conference
Kuala Lumpur, Malaysia, August 2007
Proceedings, Part III

3 Part III

Springer

Lecture Notes in Computer Science 4707

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Osvaldo Gervasi   Marina L. Gavrilova (Eds.)

Computational Science and Its Applications – ICCSA 2007

International Conference
Kuala Lumpur, Malaysia, August 26-29, 2007
Proceedings, Part III

Springer

Volume Editors

Osvaldo Gervasi
University of Perugia, Department of Mathematics and Computer Science
Via Vanvitelli, 1, 06123 Perugia, Italy
E-mail: osvaldo@unipg.it

Marina L. Gavrilova
University of Calgary, Department of Computer Science
500 University Dr. N.W., Calgary, AB, Canada
E-mail: marina@cpsc.ucalgary.ca

Associated Editors:

David Taniar
Monash University, Clayton, Australia

Andrès Iglesias
University of Cantabria, Santander, Spain

Antonio Laganà
University of Perugia, Italy

Deok-Soo Kim
Hanyang University, Seoul, Korea

Youngsong Mun
Soongsil University, Seoul, Korea

Hyunseung Choo
Sungkyunkwan University, Suwon, Korea

Library of Congress Control Number: 2007933006

CR Subject Classification (1998): F, D, G, H, I, J, C.2-3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-74482-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-74482-5 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12112180 06/3180 5 4 3 2 1 0

Table of Contents – Part III

Workshop on CAD/CAM and Web Based Collaboration (CADCAM 07)

Workshop on Component Based Software Engineering and Software Process Model (CBSE 07)

Workshop on Distributed Data and Storage System Managemnt (DDSM 07)

Workshop on Embedded Systems for Ubiquitous Computing (ESUC 07)

General Track

Defining Security Architectural Patterns Based on Viewpoints

David G. Rosado[1], Carlos Gutiérrez[2], Eduardo Fernández-Medina[1], and Mario Piattini[1]

[1] ALARCOS Research Group. Information Systems and Technologies Department UCLM-Indra. Research and Development Institute. University of Castilla-La Mancha Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain
{David.GRosado, Eduardo.Fdez-Medina, Mario.Piattini}@uclm.es
[2] Correos Telecom, Conde de Peñalver, 19 bis 6ª pl. 28006 Madrid, Spain
carlos.gutierrez@correos.es

Abstract. Recently, there has been a growing interest in identifying security patterns in software-intensive systems since they provide techniques for considering, detecting and solving security issues from the beginning of its development life-cycle. This paper describes how security architectural patterns lack of a comprehensive and complete well-structured documentation that conveys essential information of its logical structure, run-time behaviour, deployment-time and monitoring configuration, and so on. Thus we propose a set of security viewpoints to describe software-intensive security patterns adhered to ANSI/IEEE 1471-2000. In order to maximize comprehensibility, we make use of well-known language notations such as UML to represent all the necessary information for defining a software-intensive architectural security pattern conforming to the IEEE 1471-2000 standard. We investigate security architectural patterns from several IEEE 1471-2000 compliant viewpoints.

Keywords: Software Architecture**,** Security patterns, viewpoints, security.

1 Introduction

In most organizations, the importance of application-level security is often underestimated until an application faces a major security breach that causes a serious loss or downtime. Most of the time, it is clear that the probable cause of the failure is related to deficiencies in the application architecture and design, the programming, the coding security, the runtime platform, and the tools and utilities used (e.g.: COTS). The primary responsibility, of course, belongs to the application architects and developers who contributed to the application design and program code. As a result, today it is mandatory to adopt a proactive security approach during the application development life cycle that identifies critical security aspects. Architects and developers today must design security into their applications from the beginning of its lifecycle [1].

Software architecture has emerged as an important sub-discipline of software engineering, particularly in the realm of large system development. Architecture gives

O. Gervasi and M. Gavrilova (Eds.): ICCSA 2007, LNCS 4707, pp. 262 – 272, 2007.

us intellectual control over a complex system by allowing us to focus on the essential components and their interactions, rather than on extraneous details [2].

The properties that the system exhibits as it executes are among the most important issues to consider when designing, understanding, or implementing a system's architecture. What the system computes is, of course, one of these issues. But nearly as important are properties (i.e., quality attributes) such as performance, reliability, security, or modifiability. The architecture must be documented to communicate how it achieves those properties [2].

Recently, there has been a growing interest in identifying security patterns in software-intensive systems since they provide techniques for considering, detecting and solving security issues from the beginning of its development life-cycle [3][4][5][6]. Security patterns work together to form a collection of coordinated security countermeasures thereby addressing host, network and application security.

This paper describes how security architectural patterns lack of a comprehensive and complete well-structured documentation that conveys essential information of its logical structure, run-time behaviour, deployment-time and monitoring configuration, constraints, elements, and so on. In consequence, we show an alternative way for describing architectures from viewpoints and views, and therefore we can add more information about the pattern in the template used for defining patterns. Therefore we propose a set of viewpoints to define software-intensive security patterns adhered to

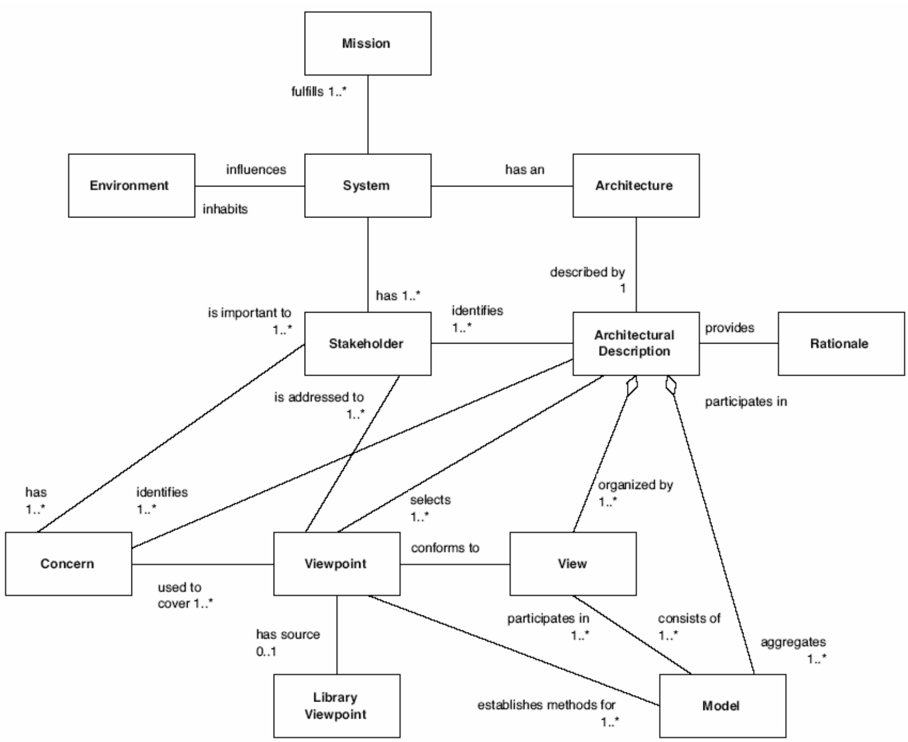

Fig. 1. Conceptual model of architectural description [7]

ANSI/IEEE 1471-2000 [7], the Recommended Practice for Architectural Description. This standard represents an emerging consensus for specifying the content of an architectural description for a software-intensive system. This approach is based on the well-known architectural concept of views [8], and holds that documentation consists of defining the relevant security views and then describing the information that applies to more than one security view. Each view conforms to a viewpoint, which in turn is a realization of the concerns of one or more stakeholders.

To put views and viewpoints in context (that we will define later in the section 2), consider the conceptual model in Fig. 1, which illustrates how views and viewpoints relate to the other important architectural concepts [9].

The remainder of this paper is organized as follows. Section 2 discusses of the importance of defining software architectures and the two most important concept associated with software architecture definition: the view and viewpoint; In section 3, we will define security patterns and what security architectural patterns are; In section 4 we will introduce the viewpoint's model to defining security patterns and we will describe the viewpoint template defined by IEEE 1471-2000 standard; In section 5 an overview of the IEEE 1471-2000 compliant Security Subsystem Design viewpoint's template definition will be shown. Finally, we will put forward our conclusions and future work.

2 Software Architecture Documentation

The architecture must be documented to communicate how it achieves the properties such as performance, reliability, security, or modifiability. Fundamentally, architecture documentation can serve three different functions [2]: a) A means of education. Typically, this means introducing people to the system. The people may be new members of the team, external analysts, or even a new architect; b) A vehicle for communication among stakeholders. A stakeholder is someone who has a vested interest in the architecture. The documentation's use as a communication vehicle will vary according to which stakeholders are communicating; c) A basis for system analysis. To support analysis, the documentation must provide the appropriate information for the particular activity being performed.

Architecture documentation must balance these varied purposes. It should be abstract enough to be quickly understood by new developers. It should be sufficiently detailed so that it serves as a blueprint for its construction. At the same time, it should have enough information so that it can serve as a basis for analysis [2].

Perhaps the most important concept associated with software architecture documentation is the view. A software architecture is a complex entity that cannot be described in a simple one-dimensional fashion.

IEEE 1471 [7] defines Architectural View as a representation of a particular system or part of a system from a particular perspective, and defines Architectural Viewpoint as a template that describes how to create and use an architectural view. A viewpoint includes a name, stakeholders, concerns addressed by the viewpoint, and the modeling and analytic conventions.

3 Security Patterns

Security patterns provide techniques for identifying and solving security issues. They work together to form a collection of best practices (to support a security strategy) and they address host, network and application security [10]. The benefits of using patterns are: they can be revisited and implemented at anytime to improve the design of a system; less experienced and non-expert security practitioners can benefit from the experience of those more fluent in security patterns; they provide a common language for discussion, testing and development; they can be easily searched, categorized and refactored; they provide reusable, repeatable and documented security practices; they do not define coding styles, programming languages or vendors [10].

Several authors (as Kienzle and Elder [11]) identify two broad categories of security patterns: i) Structural patterns that can be implemented in the final product. They encompass design patterns, such as those used by the Gang of Four. They often include diagrams of structure and descriptions of interaction; ii) Procedural patterns that can be used to improve the process for development of security-critical software. They often impact the organization or management of a development project.

Design strategies determine which application tactics or design patterns should be used for particular application security scenarios and constraints [1]. Security patterns are an abstraction of security issues (threats, attacks and vulnerabilities [12]) that address a variety of security requirements and specify the most suitable countermeasures. They can be architectural patterns that depict how a security problem can be resolved architecturally, or they can be defensive design strategies upon which secure code can be later built [1].

An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them [13]. An architectural pattern is a high-level abstraction. The choice of the architectural pattern to be used is a fundamental design decision in the development of a software system. It determines the system-wide structure and constrains the design choices available for the various subsystems. It is, in general, independent of the implementation language to be used.

There is no specific level of detail for security patterns. Different potential stakeholders of security patterns work at different levels, see different characteristics, functionalities, connections and behavior, and possess different concerns of a same pattern. Current work on defining and describing security patterns [1][5][6][14][15] do not consider all of these levels of detail. Then, we propose a set of security viewpoints, adhered to IEEE 1471-2000 standard, for defining security architectural patterns in such a way that all of the aforementioned stakeholders' issues are addressed.

4 Defining Security Viewpoints for Security Architectural Patterns

A software pattern can be described through a set of properties (a template) such as name, problem, solution, and so on. 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 [3]. Some authors [16][17] describe a template for patterns indicating the main categories and characteristics that they consider more important. There are many definitions of patterns following these templates as we can see in [18], where too it makes a comparison between security patterns.

We attempt do more extensive the template adding new information from the stakeholders' viewpoint following as reference the "4+1" view model [8], Fig. 2, where five software architecture's views (Logical, Process, Deployment, Implementation and Use-Case Views) are described.

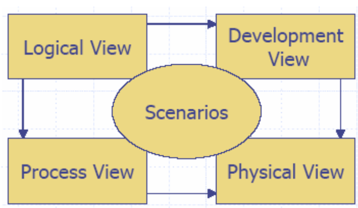

Fig. 2. "4+1" View Model

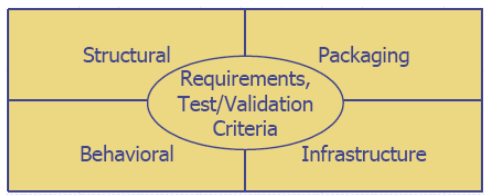

Fig. 3. Newer "4+1" View Model

Actually, the view model is changing (Fig. 3). The logical view and development view are combined into the structural view. The structural view combines both the abstract logical view and the more detailed development view. The process view has been incorporated into the behavioral view. The process view only defines units of execution where the behavioral view includes these plus important behavioral interactions between the architectural elements. The packaging view is new, as component oriented languages allow grouping of structural elements into packages. The new infrastructure view maps closely with the physical view in the older model. The Scenario view in the old model, which represented requirements now includes a logical rendering of them (typically rendered as use-cases), as well as definition of test cases needed to verify the software product.

Obviously, since the 4+1 views preceded IEEE 1471, they do not meet the definition of views as specified in the standard. The 4+1 views are more closely aligned with the concept of viewpoint as defined by IEEE 1471 standard. The 4+1 views describe a collection of representations that provide guidance for software architects. The viewpoints we discuss here are within the spirit of the 4+1 views.

ANSI/IEEE 1471-2000 [7] provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. For defining viewpoints, IEEE 1471standard defines a set of elements or sections (template) [19] that can be seen in Fig. 4.

❶ **Abstract**: this section provides a brief overview of the viewpoint;

❷ **Stakeholders and Their Concerns Addressed**: this section describes the stakeholders and their concerns that this viewpoint is intended to address;

❸ **Elements, Relations, Properties, and Constraints**: this defines the types of elements, the relations among them, the significant properties they exhibit, and the constraints they obey for views conforming to this viewpoint;

❹ **Language(s) to Model/Represent Conforming Views**: this section lists the language or languages that will be used to model or represent views conforming to this viewpoint, and cite a definition document for each;

❺ **Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria**: this section describes rules for consistency and completeness that apply to views in this viewpoint, as well as any analysis of evaluation techniques that apply to the view that can be used to predict qualities of the system whose architecture is being specified;

❻ **Viewpoint Source**: This section provides a citation for the source of this viewpoint definition, if any.

Fig. 4. Viewpoint Template adhered to IEEE 1471-2000

4.1 Viewpoints Catalogue

We are defining a library of security viewpoints that facilitates and formalizes the defining of security architectural patterns according to IEEE 1471-2000. By definition, these viewpoints are reusable for any software system, thus we can document security patterns, security architecture, software architecture, etc., based on our viewpoint's library.

A number of catalogues of viewpoints already exists, but we have found that all of them do not have security aspects and they are only applied to the development of functional requirements not being considered in the context of the security. In response, we have developed a set of viewpoints for the security architect and the security engineers, that build up and extend the "4+1" set, identified by Philippe Kruchten [8] and Nick Rozanski and Woods [9]. Our catalogue contains seven core security viewpoints: Logical, Process, Development, Physical, Deployment, Operational and Misuse Cases views. Our security viewpoints are represented in Fig. 5.

The *security logical viewpoint* describes the objects or object models within the security architecture that support security behavioral requirements. The *security process viewpoint* describes the security architecture as a logical network of secure communicating processes. This viewpoint assigns each method of the object model to a thread of execution and captures concurrency and synchronization aspects of the security design. The *security physical viewpoint* maps software onto hardware and network elements and reflects the distributed aspect of the security architecture. The *security development viewpoint* focuses on the static organization of the software in the security development environment and deals with issues of configuration

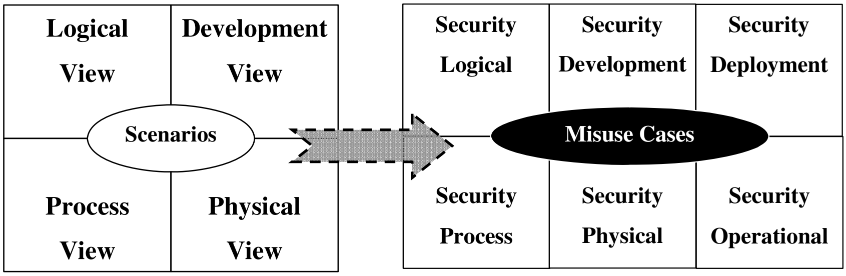

Fig. 5. From "4+1" View Model to Our Security Viewpoints

management, security development assignments, security responsibilities, and countermeasures. The *security deployment viewpoint d*escribes the security environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment. The aim of the *Security Operational viewpoint* is to identify security system-wide strategies for addressing the operational concerns of the system's stakeholders and to identify solutions that address these.

Moreover, we are defining a new viewpoint's template extending the template of IEEE 1471-2000 aforementioned and we have added new sections in the context of the security as are the follows:

- Security properties to be addressed by the security policy on the basis of the security viewpoint's elements. We consider that the complete security policy of a security pattern is the aggregation of the security policies defined for each security viewpoint.
- Security metrics to be taken into account in this viewpoint.
- Security procedures to be into taken into account from this viewpoint; for instance, from physical viewpoint, procedures to restore the physical node in which the security services defined by the pattern are running, or from logical viewpoint, how to carry out the off-line exchange of key material between the involved parties.
- Best practices: for example from developer's viewpoint, techniques for secure programming, or from physical viewpoint, topologies of secure networks.

5 Definition of Security Design Subsystem Viewpoint

Each viewpoint aforementioned can be divided in different viewpoint satisfying the interest of a particular stakeholder. A series of viewpoints is then used to elaborate the details of the general viewpoint. Selecting the security design subsystem viewpoint and considering the template aforementioned, we defined this viewpoint as presented in Fig. 6. We can say that this viewpoint locates into of the Security Development viewpoint and helps to define it.

❶ **Abstract.** This viewpoint shows security module decomposition and the use between systems of software system. Each security module interprets itself as a subsystem to develop; therefore it is an entity in construction time, that it can communicate with others security subsystems for completing its functionality. From here on a security module defined in a contextual view, we can show its decomposition in security subsystems. The decomposition continues until that each module or subsystem of security is allocated to a unique responsible of development or team.

❷ **Stakeholders and their concerns addressed**. Secure applications will be developed by (at least) three different roles:

- Application software developers that focus on the business logic (1),
- Security providers that focus on the design and implementation of reusable frameworks of security logic (2),
- Security engineers that implement the security policy for a particular application and focuse on how the system is implemented from the perspective of security, and how security affects the system properties. It examines the system to establish what information is stored and processed, how valuable it is, what threats exist, and how they can be addressed.
- Project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
- Maintainers, who are tasked with modifying the software elements;
- Testers and integrators who use the modules as their unit of work.

❸ **Elements, Relations, Properties, and Constraints**.

- Security modules are units of implementation, and its decomposition in shorter modules, just as use dependency existent between them.
- Relations between security modules can have the semantic associated 'is-part-of' or 'utilize'.
- The last level of subsystems called security design subsystems, defined in the views according to this viewpoint must:
  - To be set of products of work of design assigned to different develop teams;
  - Security subsystems will correlate with the construction directories that will be developed, tested and handed over respective teams of development;
  - Following modality origins, the security subsystems must exhibit high cohesion and low coupling;
- These subsystems will be the lower level entities for which the software architects team will need to define the interface;
- Multiple subsystems of security can be assigned to one same development team, but each security subsystem will be developed, tested and versioning of independent form;
- Each security subsystem can be considered as a system to design by the security design team to who have been assigned.

❹ **Language(s) to Model/Represent Conforming Views**.

- The representation language used will be UML and extensions for security aspects as UMLSec [20][21] and of SecureUML [22].
- Each module or subsystem of security will represent itself as a stereotyped UML packet with the reserved word <<subsystem>>. The use relations will show as relations of dependence UML including the stereotype <<uses>> and decomposition relations with nesting of UML packets.

Fig. 6. Security Design Subsystem Viewpoint

- The interfaces that implements each system are modeled as UML interfaces and the name of the service to include in each interface correspond with the names of the use cases defined in the abstraction level of “Goal Summarize” [23] for each subsystem.
- The design subsystem included into views according to this viewpoint will declare a realization of a or more interfaces whose methods correspond with use cases in the abstraction level “User Goal” specified in the model of use cases of this design subsystem.

❺ **Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria**. Revision checking with the different development groups of form that they understand the context of the subsystem that they are going to develop (what system comes from) so as the interfaces with others design subsystems. Some analysis and evaluation methods are described by Ronald Wassermann [3] and Jan Jürjens [24].

❻ **Viewpoint Source**. Viewpoint of Design Subsystem [25].

Fig. 6. *(continued)*

6 Conclusions

It is important to describe or document a software architecture because it serves to introduce people to the system, it serves as a vehicle for communication among stakeholders, and it serves as a basis for system analysis. Moreover, an architecture documented is crucial for understanding its main characteristics, its functionality, its components and connections, its behaviour, and so on.

As an architectural pattern is a micro-architecture, too it will be important to or describe its main characteristics for that stakeholder can use and analyze the pattern at the time of integrating it in the design of the application, or in the design of the whole architecture.

In this paper, we have described an architectural pattern from viewpoints attempting to give a vision wider of its main characteristics, of its design, connections, elements, interfaces, implementation, classes and behavior. We have added news sections to the existing templates, extending the information about the pattern. The enhanced security pattern template presented herein contains additional information, including behavior, constraints, and related security principles, that addresses difficulties inherent to the development of security-critical systems. The adoption of IEEE 1471 and the upcoming release of the UML 2.0, UMLSec [21] and SecureUML [22] should help improve the future practice of security software architecture.

Our intention is define security architectural patterns by means of a views template and a viewpoint template as to recommend ANSI/IEEE 1471-2000 [7], that it provides guidance for choosing the best set of views to document. We have defined a viewpoints’ catalogue and we have added and we are adding new elements or sections to the viewpoint template of IEEE 1471-2000 standard. We will create a full template of views for security architectural patterns. Our research concentrates in defining a library of viewpoints adhered to IEEE 1471-2000 which instance are the views that we can define following the documentation IEEE 1471-2000 [19]. Of this form we could have a library of viewpoints to define security architectural patterns.

Acknowledgements

This research is part of the following projects: DIMENSIONS (PBC-05-012-2) and MISTICO (PBC-06-0082) both partiality supported by FEDER and by the "Consejería de Educación y Ciencia de la Junta de Comunidades de Castilla-La Mancha" (Spain), RETISTRUST (TIN2006-26885-E) and ESFINGE (TIN2006-15175-CO5-05) granted by the "Dirección General de Investigación del Ministerio de Educación y Ciencia" (Spain).

References

1. Steel, C., Nagappan, R., Lai, R.: Core Security Patterns, p. 1088. Prentice Hall, Englewood Cliffs (2005)
2. Bachmann, F., Bass, L., Clements, P., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures:Organization of Documentation Package, Software Engineering Institute (2001)
3. Cheng, B.H.C., Konrad, S., Campbell, L.A., Wassermann, R.: Using Security Patterns to Model and Analyze Security Requirements. Monterey Bay, CA, USA, pp. 13–22 (2003)
4. Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F.: Security Patterns, 1st edn., p. 512. John Wiley & Sons, Chichester (2005)
5. Schumacher, M., Roedig, U.: Security Engineering with Patterns. In: 8th Conference on Patterns Lnaguages of Programs, PLoP 2001, Monticello, Illinois, USA (2001)
6. Yoder, J., Barcalow, J.: Architectural Patterns for Enabling Application Security, Monticello, Illinois, USA (1997)
7. IEEE, Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000). Institute of Electrical and Electronics Engineers: New York, NY, p. 29 (2000), http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html
8. Kruchten, P.: Architectural Blueprints - The "4+1" View Model of Software Architecture. IEEE Software 12(6), 42–50 (1995)
9. Rozanski, N., Woods, E.i.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, 1st edn., p. 576. Addison Wesley, Reading (2005)
10. Berry, C.A., Carnell, J., Juric, M.B., Kunnumpurath, M.M., Nashi, N., Romanosky, S.: Patterns Applied to Manage Security, in J2EE Design Patterns Applied, Ch. 5 (2002)
11. Kienzle, D.M., Elder, M.C.: Final Technical Report: Security Patterns for web Application Development (2005)
12. Firesmith, D.G.: Commom Concepts Underlying Safety, Security, and Survivability Engineering CMU/SEI-2003-TN-033. SEI (2003)
13. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, p. 476. John Wiley & Sons, Chichester (1996)
14. Fernandez, E.B., Pan, R.: A pattern language for security models. In: 8th Conference on Pattern Languages of Programs, PLoP 2001. Allerton Park, Illinois, USA (2001)

15. Security Design Patterns (2001), Available on: http://www.cgisecurity.com/lib/security-DesignPatterns.html
16. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1994)
17. AGCS, AG Communication System. Template Pattern (1996)
18. Rosado, D.G., GutiÃ©rrez, C., Fernandez-Medina, E., Piattini, M.: A Study of Security Architectural Patterns, Vienna, Austria, pp. 358–365. IEEE Computer Society, Los Alamitos (2006)
19. Software Architecture Document (SAD) (2006) Available on: www.sei.cmu.edu/architecture/SAD_template2.dot
20. Jurjens, J.: Towards Secure Systems Development with UMLsec. In: Hussmann, H. (ed.) ETAPS 2001 and FASE 2001. LNCS, vol. 2029, pp. 187–200. Springer, Heidelberg (2001)
21. Jurjens, J.: UMLsec: Extending UML for Secure Systems Development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)
22. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-Based Modeling Language for Model-Driven Security. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
23. CockBurn, A.: Writing Effective Use Cases, p. 270. Addison-Wesley Professional, Reading (2000)
24. Deubler, M., Grünbauer, J., Jürjens, J., Wimmel, G.: Sound Development of Secure Service-based Systems. In: Second International Conference on Service Oriented Computing (ICSOC), ACM Press, New York (2004)
25. Garlan, J., Anthony, R.: Large-Scale Software Architecture, p. 278. John Wiley & Sons, Chichester (2002)