A UML-Based Methodology for Secure Systems: The Design Stage

Eduardo B. Fernandez, Tami Sorgente, and María M. Larrondo-Petrie

Department of Computer Science and Engineering, Florida Atlantic University, 777 Glades Road SE-300, Boca Raton, Florida 33431-0991 USA

Abstract. We have previously proposed a UML-based secure systems development methodology that uses patterns and architectural layers. We studied requirements and analysis aspects and combined analysis patterns with security patterns to build secure conceptual models. Here we extend this methodology to the design stage. Design artifacts provide a way to enforce security constraints. We consider the use of views, components, and distribution.

1 Introduction

In a previous paper we proposed a new type of analysis pattern, called a *Semantic Analysis Pattern (SAP)* [1]. A Semantic Analysis Pattern is a pattern that describes a small set of coherent use cases that together describe a basic generic application. The use cases are selected in such a way that the application can fit a variety of situations. Using SAPs we developed a methodology to build the conceptual model systematically. To use the methodology it is necessary to first have a good collection of patterns. We have developed several analysis patterns, e.g., [2], [3], [4], [5], [6], and a good number of others exist in the literature, e.g., [7], [8]. These patterns are abstractions of real applications. We have made some experiments to show that they result in good quality conceptual models that are convenient to use when developing complex applications. An important quality aspect is security.

Security is a serious problem, especially for complex applications. This complexity could provide many opportunities for attacks. It is then important to build these applications in a systematic way. We have proposed a development approach that applies security throughout the whole lifecycle and uses security patterns [9]. As part of this work we have produced a variety of security patterns, e.g., [10], [11]. We showed that we can combine SAPs and security patterns in a natural way to create authorized SAPs, which can be converted into secure designs [12]. We start by using SAPs to build conceptual models and the necessary security constraints are then defined. We consider here how to carry over the security model of the analysis stage into the design stage. Design artifacts such as views, components, and distribution can be used to enforce the security constraints defined in the conceptual model.

B. Fernandez E., Sorgente T. and M. Larrondo-Petrie M. (2005).
A UML-Based Methodology for Secure Systems: The Design Stage.
In *Proceedings of the 3rd International Workshop on Security in Information Systems*, pages 207-216
DOI: 10.5220/0002558102070216

Section 2 introduces SAPs and security patterns. Section 3 shows how security patterns are added to conceptual models. In Section 4 we show how these analysis models are converted into design models. We end with some conclusions.

2 SAPs and security patterns

The development of object-oriented software starts from requirements normally expressed as use cases. The requirements are then converted during the analysis stage into a conceptual or domain model. Analysis is a fundamental stage because the conceptual model can be shown to satisfy the requirements and becomes the skeleton on which the complete system is built. No good design or correct implementation is possible without good analysis, the best programmers cannot make up for conceptual errors. In addition, the correction of analysis errors becomes very expensive when these errors are caught in the code.

An instance of a SAP is produced in the usual way: use cases, class and dynamic diagrams, etc. (See [13]). We select the use cases in such a way that they include aspects that may be common to many applications. We can then generalize the original pattern by abstracting its components and later we may derive new patterns from the abstract pattern by specializing it. We can also use analogy to directly apply the original pattern to a different situation. As indicated earlier, we have developed secure patterns of this type, one of which, the Patient Treatment Record, we use here to illustrate our ideas.

The Patient Treatment Record Pattern describes the treatment or stay instance of a patient in a hospital [5]. The hospital may be a member of a medical group. Each patient has a primary physician, an employee of the hospital. Upon admission the patient record is created or information is updated from previous visit(s). Inpatients are assigned a location, nurse team and consulting doctors. This pattern realizes use cases Admit Patient, Discharge Patient, Assign Assets to an Inpatient, and Assign Nurse to a Location. Assets of the medical group are assigned to a patient through associations. Figure 1 shows associations between classes **Doctor**, **Nurse,** and **Location** and class **Patient**, which describe the corresponding assignments. In particular, all patients are assigned a primary doctor while inpatients may also be assigned consulting doctors. Locations include the room assigned to an inpatient or other places for specific treatments. The assets of the medical group are organized in a hierarchical arrangement that describes their physical or administrative structure. Specifically, **MedicalGroup** includes some **Hospitals,** and in turn each hospital includes some **Buildings**. Each treatment **Location** is part of a building. The class **Employee** classifies the types of personnel that are assigned to patients.

One of the most basic security patterns is the Role-Based Access Control (RBAC) pattern [10]. In this model users join roles according to their tasks or jobs, and rights are assigned to the roles. In this way a need-to-know policy can be applied, where roles get only the rights they need to perform their tasks. Use cases can be used as references to define the needed rights for each role [14]. Figure 2 shows the class diagram for this pattern. Classes **Role**, **ProtectionObject,** and **Right** define the authorizations for roles. A right defines an access type indicating in what manner the role can access the object. An operation checkRights can be used to find the rights of a particular role or which roles can access a given object.

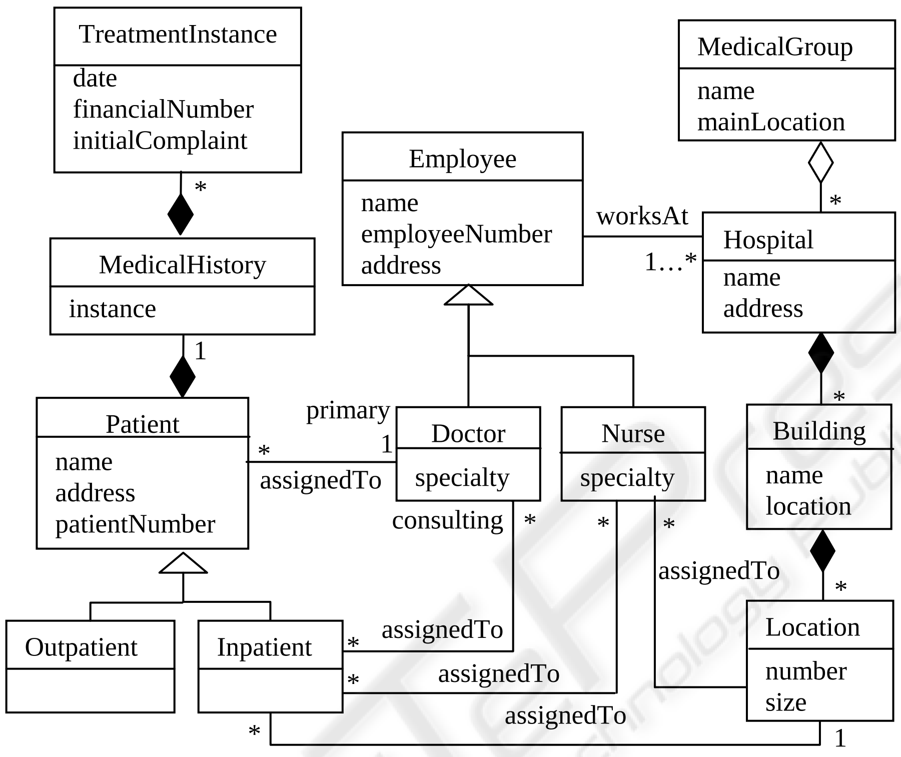

Fig. 1. Class diagram for Patient Treatment Record pattern

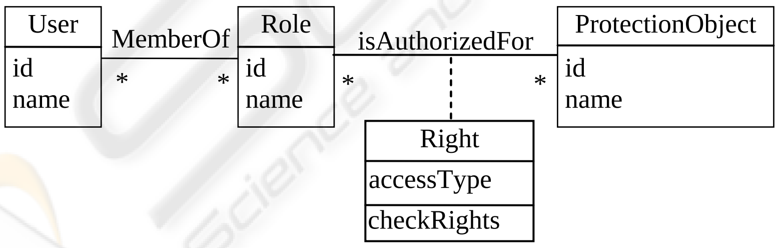

Fig. 2. The Role-Based Access Control pattern

3 The analysis stage

The use cases define all the ways to use the system and we need to give the involved actors rights to perform their interactions [14]. Figure 3 shows a sequence diagram that implements the use case Admit Patient when we have a new patient. The administrative clerk needs rights to define a guardian and to create a patient record, patient information, a medical history, and a treatment instance (these are implied by the right admitPatient). She also has the right to assign assets to them. Because actors correspond to roles in a RBAC model, the rights from Figure 3 are defined in terms of roles. In Figure 4 we have added authorization rules to perform these functions to the Patient Treatment pattern. This is performed by adding instances of the RBAC pattern.

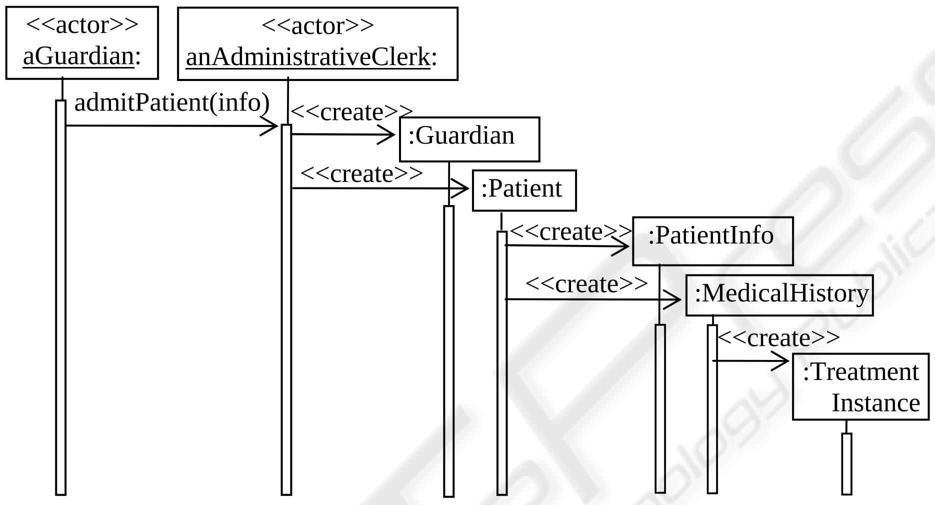

Fig. 3. Sequence diagram to admit a new patient

4 The design stage

We can now carry over the security architecture of the analysis stage to the design stage. The authorization constraints defined by the authorized SAPs must be reflected into specific authorizations in the design artifacts, e.g. in user interfaces, components, and others. The design stage corresponds to the definition of software layers that implement the conceptual modeling. We need to constrain their access according to the restrictions defined in the conceptual model.

For example, user interfaces can be implemented by a Model View Controller (MVC) pattern [15]. Each View corresponds to an interface for a use case and we can enforce role rights at these interfaces. Figure 5 implements the use case Admit Patient and shows the AdministrativeClerk role as the only role with the ability to admit patients and perform the required actions. A model like this can be made more specific by specializing it for a particular language. For example, it could be tailored for Java and J2EE components by using classes Observable (instead of Model), Observer, and Controller from the Java libraries.

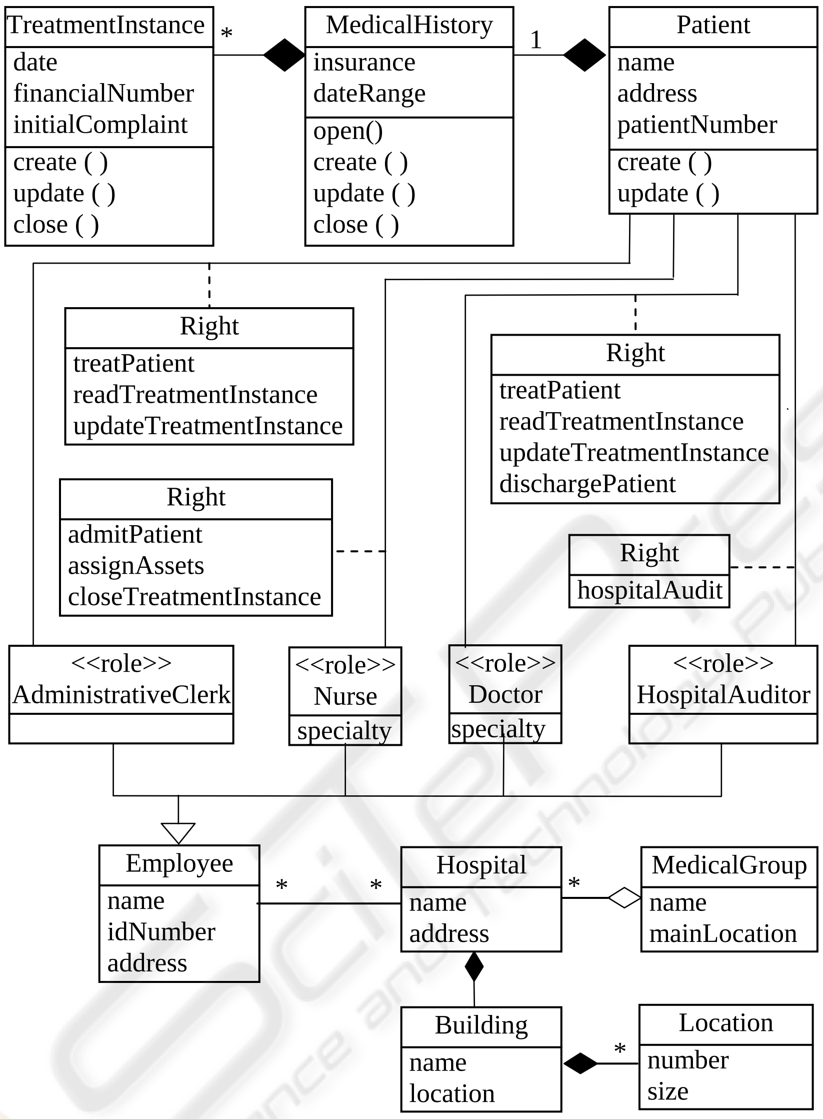

Fig. 4. Patient treatment pattern with RBAC authorizations

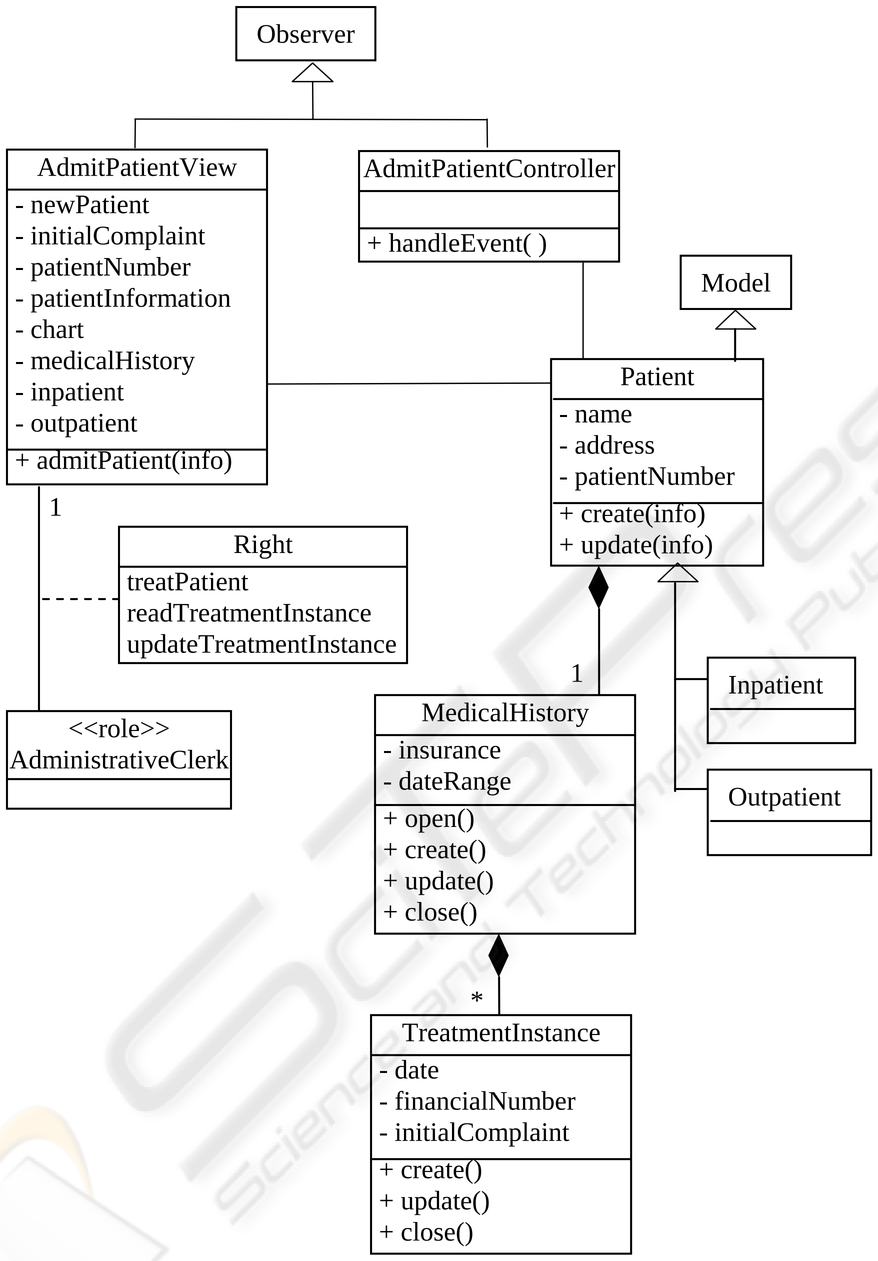

Fig. 5. Adding security enforcement through interfaces

It is also necessary to define rights in J2EE or .NET components. This security is specified in their deployment descriptor that is written in XML [16]. Security in J2EE is based on roles and matches well the model we are using. For example, if the Patient class is implemented as a component, its descriptor may specify that TreatmentInstance can only be modified by doctors. This rule is at a lower level than interface rules and it could be considered more fundamental, and it could not be overridden, i.e. no rule in the AdmitPatient View can give somebody who is not a doctor the right to modify patient treatment instances. This approach adds a second line of defense against administrator errors (the Defense in Depth principle). Similarly, components can access persistent data in relational databases using JDBC. These relations could include further authorizations to provide another line of defense. When we do this, it is necessary to make sure that the rights defined in the views, components, and database items do not conflict with each other. To determine possible overlappings we need to map security constraints across architectural levels [17] [18].

Distribution is usually performed through two basic approaches:

- Distribution of objects using an Object Request Broker, e.g., CORBA, DCOM, .NET Remoting. We can add security rules to the broker pattern to control access to remote objects.
- Distribution of component and interfaces, e.g. web services. We can control access to web services using an XML firewall [11].

Since distribution provides another place to perform access control it needs again to be coordinated with the other authorizations. There are some interesting mapping problems to study here.

5 Discussion

The steps discussed above are part of our methodology. In past work we have shown how to derive rights for roles from use cases [14]. We have also shown the need to relate attacks to use cases [9]. Another aspect of our work discussed how to create secure conceptual models by combining SAPs and security patterns [12]. We showed an example of that approach in Section 3, and Figure 6 shows a secure financial model that applies analogy to the medical example. A related idea makes use of aspects [19] [20]. We have also shown how UML can be used to represent any of the existing access control models as well as sets of security constraints that may not follow any model [21]. SecureUML [22] uses RBAC as a metamodel for specifying and enforcing security. They make use of Model-Driven-Architecture (MDA) [23] to generate secure code. However, they do not consider the effect of any design constructs, they enforce constraints directly in the code. We consider explicitly the details of the lower levels. The hierarchy of models that we use is still appropriate for the use of MDA to generate automatically some aspects of the lower levels models. Other approaches based on UML do not provide a complete lifecycle software development but focus only on specific steps. Mouratidis and Giorgini have developed an approach to secure software development that is also applied through all stages [24]. Their approach uses a special methodology, Tropos, based on agents, and focuses more on security requirements.

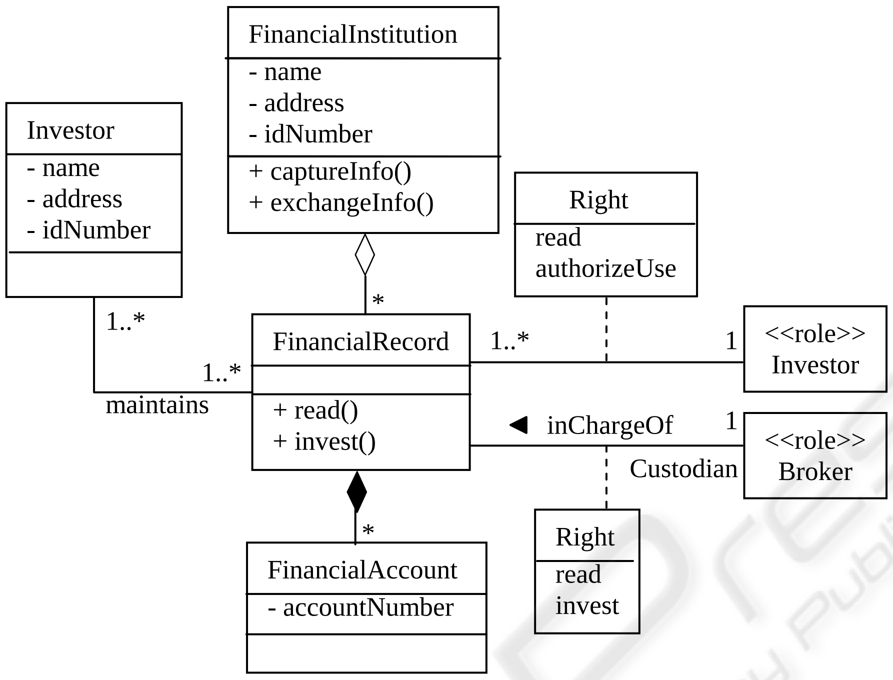

Fig. 6. Financial pattern with RBAC

6 Conclusions

A good analysis model for a portion of a complex system can be abstracted and become an analysis pattern that can be used in other applications. Their use can save time and improve the quality of a system. An important advantage of SAPs is that they can be combined easily with security patterns, resulting in authorized applications. The security defined in the conceptual model can be enforced in the design model using security patterns at the lower architectural levels, including user interfaces, components, distribution, and database adapters. We are currently developing patterns for secure Brokers and for web services. As far as we know this is the first time a secure methodology is applied at the design stage carrying a secure model from the analysis stage and considering the effect of the lower architectural levels.

Acknowledgements

This work was supported by a grant from DISA, administered by Pragmatics, Inc.

References

1. Fernandez, E. B., and Yuan, X.: Semantic analysis patterns. In: Proceedings of 19th International Conference on Conceptual Modeling, (2000) 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf
2. Fernandez, E. B., and Yuan, X.: An analysis pattern for reservation and use of entities. In: Proceedings of the Pattern Languages of Programs Conference, PLoP99 (1999). http://st-www.cs.uiuc.edu/~plop/plop99
3. Fernandez,E. B., Yuan, X., and Brey, S.: Analysis Patterns for the Order and Shipment of a Product. In: Proceeding of the Pattern Languages of Programs Conference, PLoP00, (2000). http://hillside.net/plop/2000/
4. Fernandez, E. B., and Yuan, X.: An Analysis Pattern for Repair of an Entity. In: Proceedings of the Pattern Languages of Programs Conference, PLoP01 (2001). http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions
5. Sorgente, T., and Fernandez, E. B.: Analysis patterns for patient treatment. In: Proceedings of the Pattern Languages of Programs Conference, PLoP04 (2004). http://jerry.cs.uiuc.edu/~plop/plop2004/accepted_submissions
6. Yuan, X., and Fernandez, E. B.: An analysis pattern for course management. In: Proceedings of the Pattern Languages of Programs Conference, PLoP03 (2003). http://hillside.net/europlop
7. Fowler, M.: Analysis patterns – Reusable object models, Addison-Wesley (1997).
8. Hamza, H. S. and Fayad, M. E.: The Negotiation Analysis Pattern. In: Proceedings of the Pattern Languages of Programs Conference, PLoP04 (2004). http://hillside.net/plop/2004/
9. Fernandez, E. B.: A methodology for secure software design. In: Proceedings of the 2004 Intl. Symposium on Web Services and Applications, ISWS'04, Las Vegas, Nevada, 21-24 June 2004 (2004).
10. Fernandez, E. B., and Pan, R.: A Pattern Language for security models. In: Proceedings of the Pattern Languages of Programs Conference, PLoP01 (2001). http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions
11. Delessy-Gassant, N., Fernandez, E.B., Rajput, S., and Larrondo-Petrie, M. M: "Patterns for application firewalls. In: Proceedings of the Pattern Languages of Programs Conference (PLoP2004). http://hillside.net/plop/2004/
12. Fernandez, E. B.: Layers and non-functional patterns. In: Proceedings of ChiliPLoP03, Phoenix, Arizona, 10-15March 2003 (2003). http://hillside.net/chiliplop/2003/
13. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd edition, Prentice-Hall (2005).
14. Fernandez, E. B., and Hawkins, J. C.: Determining Role Rights from Use Cases. In: Proceedings of the 2nd ACM Workshop on Role-Based Access Control, ACM (1997) 121-125. http://www.cse.fau.edu/~ed/RBAC.pdf
15. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, Vol. 1, Wiley (1996).
16. Koved, L., Nadalin, A., Nagarathan, N., Pistoia, M., and Schrader, T.: Security challenges for Enterprise Java in an e-business environment. In: IBM Systems Journal, Vol. 40, No. 1, (2001), 130-152.
17. Fernandez, E. B.: Coordination of security levels for Internet architectures. In: Proceedings of the 10th International Workshop on Database and Expert Systems Applications (1999) 837-841. http://www.cse.fau.edu/~ed/Coordinationsecurity4.pdf
18. Wood, C. Summers, R. C. and Fernandez, E. B.: Authorization in multilevel database models. In: Information Systems, Vol. 4 (1979) 155-161.

19. Georg, G., France, R., and Ray, I.: Creating Security Mechanism Aspect Models from Abstract Security Aspect Models. In: Workshop on Critical Systems Development with UML, UML2003, October 2003 (2003) http://www.cs.colostate.edu/~georg/aspectsPub/CSDUML03.pdf
20. Ray, I., France, R. B., Li, N., and Georg, G.: An Aspect-Based Approach to Modeling Access Control Concerns. In: Journal of Information and Software Technology, Vol, 46, No. 9, July 2004, (2004) 575-587, http://www.cs.colostate.edu/~georg/aspectsPub/IST04.pdf
21. Fernandez, E. B., Larrondo-Petrie, M. M., Sorgente, T., Rajput, S., and VanHilst, M.: UML-based access control models. Submitted for publication.
22. Lodderstedt, T., Basin, D. A., and Doser, J.: SecureUML: A UML-based modeling language for model-driven security. In: Proceedings of the 5th International Conference on UML, UML 2002, Lecture Notes in Computer Science, Vol. 2460, Springer-Verlag, Berlin Heidelberg New York (2002) 426-441.
23. Object Management Group. http://www.omg.org/uml
24. Mouratidis, H., and Giorgini, P.: Analyzing security in information systems. In: Proceedings of the 2nd International Workshop on Security and Information Systems, WOSIS 2004, Porto, Portugal (2004).