Patterns and Pattern Diagrams for Access Control

Eduardo B. Fernandez[1] , Günther Pernul[2] and Maria M. Larrondo-Petrie[1]

[1] Florida Atlantic University, Boca Raton, FL 33431, USA
ed@cse.fau.edu | petrie@fau.edu
[2] University of Regensburg, Universitätsstraße 31, Regensburg, Germany
guenther.pernul@wiwi.uni-regensburg.de

Abstract: Access control is a fundamental aspect of security. There are many variations of the basic access control models and it is confusing for a software developer to select an appropriate model for her application. The result in practice is that only basic models are used and the power of more advanced models is thus lost. We try to clarify this panorama here through the use of patterns. In particular, we use pattern diagrams to navigate the pattern space. A pattern diagram shows relationships between patterns and we can see how different models relate to each other. A subproduct of our work is the analysis of which patterns are available for use and which need to be written. Pattern maps are also useful to perform semi-automatic model transformations as required for Model-Driven Development (MDD). The idea is to provide the designer of a secure system with a navigation tool that she can use to select an appropriate pattern from a catalog of security patterns. We also indicate how to compose new access control models by adding features to an existing pattern and how to define patterns by analogy.

1 Introduction

The development of secure systems requires that security be considered at all stages of design, so as to not only satisfy their functional specifications but also satisfy security requirements. Several methodologies that apply security at all stages have been proposed [1], [2], [3]. Some of these methodologies start from use cases and from them a conceptual model is developed. Security constraints are then defined in the conceptual model. To do this we need high-level models that represent the security policies that constrain applications. One of the most fundamental aspects of security is access control.

Although there are only a few basic access control models, many varieties of them have been proposed. It is confusing for a software developer to select an appropriate model for her application. Access control models generally represent a few types of security policies, e.g. “rights are assigned to roles”, and provide a formalization of these policies using some ad hoc notation. Four basic access control models are commonly used and they may be extended to include content and context-based access control, delegation of rights, hierarchical structuring of subjects (including roles), objects, or access types [4], temporal constraints, etc. Access control models can be defined for different architectural levels, including application, database systems, operating systems, and firewalls [5]. Some of them apply to any type of systems while some are specialized, e.g. for distributed systems.

Access control models fall into two basic categories: Mandatory Access Control (MAC), where users' rights are defined by administrators and data may be labeled to indicate its sensitivity, and Discretionary Access Control (DAC), where users may administer the data items they create and own. In a MAC model, users and data are classified by administrators and the system applies a set of built-in rules that users cannot circumvent. In a DAC model, there is no clear separation of use and administration; users can be owners of the data they create and act as their administrators. Orthogonal to this classification, there are several models for access control to information that differ on how they define and enforce their policies [6], [7]. The most common are:

- The *Multilevel model* organizes the data using security levels. This model is usually implemented as a mandatory model where its entities are labeled indicating their levels. This model is able to reach a high degree of security, although it can be too rigid for some applications. Usually, it is not possible to structure the variety of entities involved in complex applications into strictly hierarchical structures.
- The *Access Matrix* describes access by subjects (actors, entities) to protected objects (data/resources) in specific ways (access types) [8], [6], [7]. It is more flexible than the multilevel model and it can be made even more flexible and precise using predicates and other extensions. However, it is intrinsically a discretionary model in which users own the data objects and may grant access to other subjects. It is not clear who owns the medical or financial information and the discretionary property reduces security. This model is usually implemented using Access Control Lists (lists of the subjects that can access a given object) or Capabilities (tickets that allow a process to access some objects).
- *Role-Based Access Control* (RBAC), collects users into roles based on their tasks or functions and assigns rights to each role [9]. Some of these models, e.g. [10], [11], have their roles structured as hierarchies, which may simplify administration. RBAC has been extended and combined in many ways.
- *Attribute-Based Access Control* (ABAC). This model controls access based on properties (attributes) of subjects or objects. It is used in environments where subjects may not be pre-registered [12].

While these basic models may be useful for specific domains or applications, they are not flexible enough for the full range of policies present in some of these applications [5], [4]. This is manifested in the large variety of ad hoc RBAC variations that have been proposed; most of which add specialized policies to a basic RBAC model. For example, some models have added content or context-dependent access [13], delegation [14], task-based access [15], and relationships between role entities [16]. All these models effectively incorporate a set of built-in access control policies and cannot handle situations not considered by these policies, which means that a complex system may need several of these models for specific users or divisions.

All these models present a bewildering set of options to the designer, who has problems deciding which model to use. The result in practice is that only basic models are used and the power of more advanced models is thus lost. We try to clarify this panorama here through the use of patterns. In particular, we use pattern diagrams to navigate the pattern space. A pattern diagram shows relationships between patterns

(represented by rectangles with rounded corners). A subproduct of our work is the analysis of which patterns are available for use and which need to be written. Pattern maps are also useful to perform semi-automatic model transformations as required for Model-Driven Development (MDD). They can serve as metamodels of possible solutions being added at each transformation.

A pattern is an encapsulated solution to a recurrent problem in a given context. In particular, a security pattern describes a mechanism that is a solution to the problem of controlling a set of specific threats [17]. This solution is affected by some forces and can be expressed using UML class, sequence, state, and activity diagrams. A set of consequences indicate how well the forces were satisfied; in particular, how well the attacks can be handled by the pattern. A study of the forces and consequences of a pattern is important before their final adoption; however, a good initial pattern selection is fundamental to avoid a lengthy search through textual pattern descriptions. A requirement for a pattern is that the solution it describes has been used in at least three real systems [18], [19]. This is consistent with the idea of patterns as best practices. However, a pattern can also describe solutions that have not been used (or have been used only once) but appear general and useful for several situations. Because of this, we include here both types: good practices patterns and useful solutions patterns. In fact, as mentioned above, many models have never been used in practice.

We do not attempt to be exhaustive because the quantity of models is too large, some are just simple variations of others, and some appear to have scarce practical value. How exhaustive the catalog needs to be depends on the variety of applications to be handled. The idea is to provide the designer of a secure system with a way to organize a navigation tool that she can use to select an appropriate pattern from a catalog of security patterns. We also indicate how to compose new access control models by adding features to an existing pattern and how to define patterns by analogy.

Section 2 presents the use of pattern diagrams to relate access control models. Section 3 discusses how patterns can be defined at different levels of abstraction. Section 4 considers how to grow new models from existing ones, while Section 5 shows how to obtain models by analogy. We end with some conclusions.

2 Pattern diagrams for access control patterns

Access control models have two aspects: a definition of a set of rules specifying the valid accesses (some of them may be implied by other rules), and an enforcement mechanism that intercepts access requests from users or processes and determines if the request is valid. The main difference between models is on the way they define their rules, so it makes sense to separate the patterns for enforcement mechanisms; that is, we should provide separate patterns for rules and for enforcement mechanisms. Typically, there is much less variety in the enforcement mechanism: it intercepts requests and makes a decision based on the rules. As an illustration of how pattern diagrams can put models in perspective, Figure 1 shows some variations of access control models. One of the first access control models was the access matrix. The basic access matrix [7] included the tuple {s,o,t}, where s indicates a subject or active entity, o is the protected object or resource, and t indicates the type of access

permitted. [8] proved security properties of this model using the so-called HRU (Harrison-Ruzzo-Ullman) model. In that model users are allowed to delegate their rights (discretionary property, delegatable authorization), implying a tuple {s,o,t,f}, where f is a Boolean copy flag indicating if the right is allowed to be delegated or not. A predicate was added later to the basic rule to allow content-based authorization [20], becoming {s,o,t,p,f}, where p is the predicate (the predicate could also include environment variables). Patterns for the basic rule and for the one with tuple {s,o,t,p,f} were given in [21], [17]. The rule could also include the concept of Authorizer (a), becoming {a,s,o,t,p,f} [22] (Explicitly Granted Authorization). RBAC [9] can be considered a special interpretation of the basic authorization model, where subjects are roles instead of individual users. We presented two varieties of RBAC patterns in [21] and [17]. We combined it with sessions in [23] (The double-lined patterns of Figure 1). Several variations and extensions of these models have appeared. We presented a variation called Metadata-Based Access Control, which later we renamed Attribute-Based Access Control (ABAC) [12].

Figure 1 assumed that we started from some known models. Figure 2 starts from the basic components of access control to provide a more general approach to developing access control models (we did not show the labels of the links for simplicity). This diagram can be the starting point that allows a designer to select the type of access control he needs in his application. Once this abstract level is clear, we need to go to a software-oriented level where we can choose more specific approaches. The center of this diagram is Policy-Based Access Control (PABC) which indicates that the rules represent access policies, which are in turn defined by a Policy pattern. The Policy-Based Access Control pattern decides if a subject is authorized to access an object according to policies defined in a central policy repository. The enforcement of these policies is defined by a Reference Monitor pattern. Depending on its administration, PABC can be MAC or DAC. XACML is a type of PBAC oriented SOA [24], shown here as two patterns for its aspects of rule definition and evaluation. Policies can be implemented as Access Control Lists (ACLs) or Capabilities. The NIST pattern is a variety of RBAC discussed in Section 4. The reference Monitor may use a Policy Enforcement Point (PEP), a Policy Definition Point (PDP), and other patterns to describe the administrative structure of enforcement [24]. The Access Matrix can be extended with predicates or a copy flag and both can be used in another variation of this model.

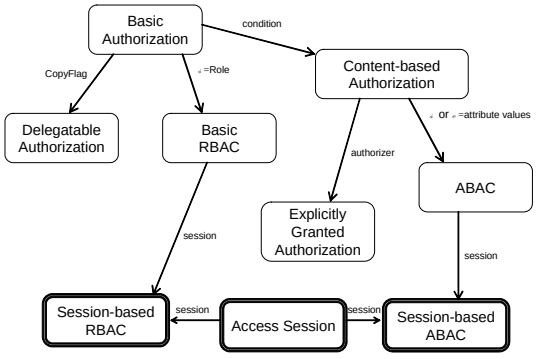

Fig 1. Relationships between access control patterns

3 Levels of abstraction

Models and their patterns may correspond to different abstraction levels; for example, the concept of session is a concept at a lower level than RBAC because it indicates an abstract mechanism to restrict the concurrent use of some rights (which may produce conflicts of interest). It is possible to make explicit in the pattern diagram the abstraction level of the patterns Figure 3 shows these levels explicitly, showing the Policy Model, Policy Rule,and reference Monitor at the highest level, while PBAC and the Session-Based Reference Monitor are more implementation-oriented. Many times we do not emphasize this separation; however, when we deal with different architectural levels this separation is important; for example, the implementation of access control at the file system level is quite different from authorization rules at the application level. Figure 3 also shows how some of the patterns of Figure 1 could have been found starting from the components of Figure 2: We can define a Session-Based Reference Monitor that requires the concept of Access Session to delimit the rights of the user. This figure also emphasizes the separation of Policy Model and Policy Rules, not shown in Figure 2.

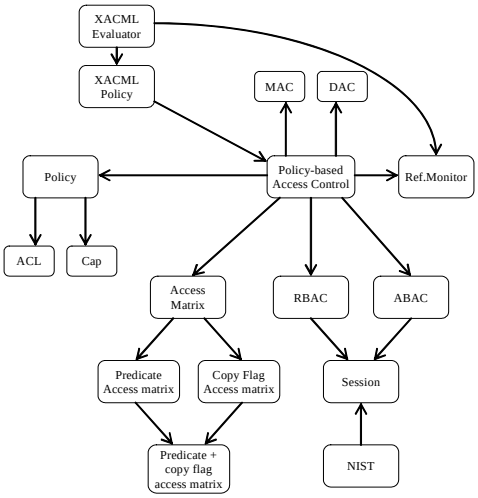

Fig. 2. A classification of access control patterns

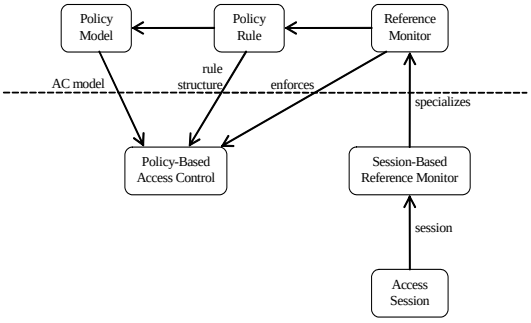

Fig. 3. Deriving specialized patterns from abstract models

4 Using the diagrams

Let's consider a financial application. The threats to this system have been identified and indicate that we need access control for classes Account, Transaction, and Customer. The designer refers to a diagram such as Figure 2 and starts by choosing PBAC because we need to apply banking policies to control access; for example, the owner of the account has the right to deposit and withdraw money from the account. The designer then chooses the Access Matrix model because access to accounts is given to individuals, not roles. As owners should only access their own accounts, we need predicates in the Access Matrix. Later, when the software architecture is defined, the designer decides to use web services, because ATMs and branch offices make this application highly distributed. Since any PBAC can be implemented using XACML, the designer implements a Predicate Access Matrix using XACML.

5 Growing new models

To apply this design approach we need good pattern catalogs. In this and the next section we see two approaches to develop catalogs. Each pattern can be augmented with new features to produce a new model with larger functionality. Figure 4 shows the basic RBAC pattern, where users are assigned to roles and roles are given rights. The National Institute of Standards and Technology (NIST) RBAC pattern follows the NIST standard [21] and allows access to resources based on the role of the subject and adds several functions to the basic model. The NIST standard also adds a linear role hierarchy. A subject may have several roles. Each role collects the rights that a user can activate at a given moment (execution context), while a session controls the way of using roles and can enforce role exclusion at execution time. Strictly, the NIST standard allows only linear role hierarchies, while Figure 5 shows recursive hierarchies (described by the Composite pattern [19]). Figure 5 shows also object/resource recursive hierarchies. Similarly, we can separate administrative roles and rights and give roles to groups of users for example. The idea is that new functions require only adding more classes and their corresponding associations. In fact, the pattern chains in the models of Figures 1 and 2 can be obtained in this way; for example, in Figure 2 we added a session to RBAC to obtain an NIST RBAC pattern.

Combining this idea with architectural levels we can define different variations of these patterns intended for more specialized uses. For example, [25] shows an RBAC model where access to objects is performed through views. Specific views carry sets of rights as in the case of database systems. We can also formalize these models by adding OCL constraints in each pattern. The constraints may make functionality more specific, for example, by defining constraints among variables, or may define pre- or post-conditions for the pattern. As each model is derived from a simpler model, it is easy for the designer to see the differences and possibilities of each model.

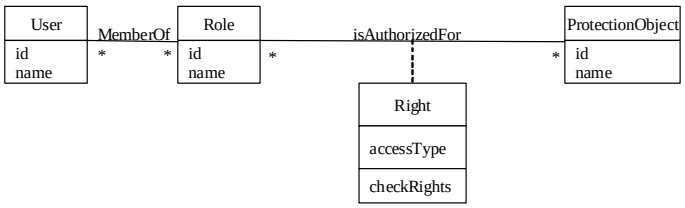

Fig. 4. Basic RBAC pattern

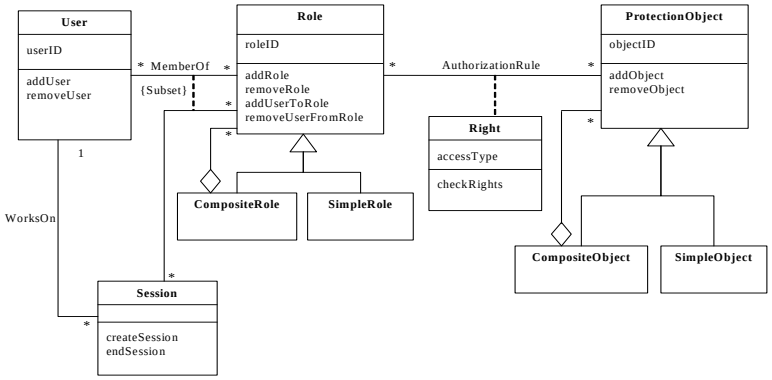

Fig. 5. Class model for RBAC with sessions and role/resource hierarchies

6 Finding access control models by analogy

Analogy is another way to derive models from existing models [26].A model for medical records such as the one in Figure 6 represents typical policies used in HIPAA and other regulations. In particular, this model represents the policies:

- A **Patient** role that has the rights to read his own record and authorize the use of this record.
- A **Doctor** role showing that a given doctor may act as custodian for a patient record.
- The **Medical Record**, that includes the constraint that any reading of a record by persons other than health providers, must be notified to the corresponding patient.
- Specific medical records may be associated (linked) with other records to describe, for example, family relationships and physical contact, needed to trace genetic or infectious diseases.

If we need a model for the Sarbanes/Oxley regulations, we can make the following analogies: Patient—Investor; Doctor—Broker; Hospital—Financial Institution; Medical Record—Financial Record. This leads to Figure 7, which is basically the same structure although the behavior semantics of some classes may be different.

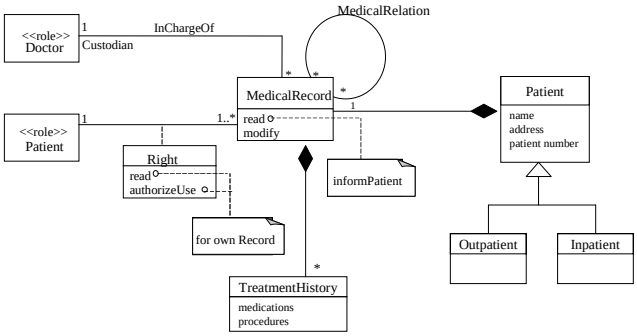

Fig. 6. A model for medical policies

6 Conclusions

We have tried to unify the many access control model varieties by using patterns. We believe that this perspective can help developers to align their needs with the selection of appropriate access control models. The selected access control pattern not only guides the conceptual security of the application but later it also guides the actual implementation of the model.

We can navigate the pattern diagram because (as shown in Section 4) patterns are composable with features, i.e. adding a feature (perhaps embodied by another pattern) produces a new pattern with extra features. If we have a pattern diagram we can navigate in it to find an appropriate pattern

Using this approach we can not only clarify the relationships of access control models but it has led us also to discover the need for new security patterns: Subject, Object, Labeled Security, DAC, MAC. Access control patterns give us also the possibility of evaluating commercial products: we can see if the product contains the corresponding pattern in its design.

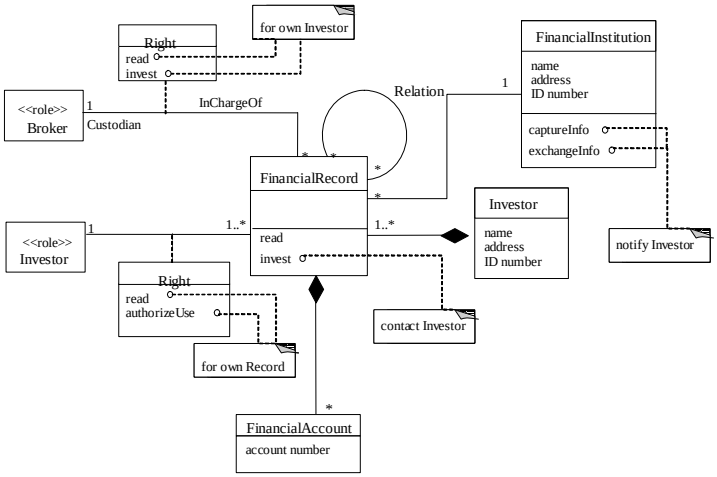

Fig. 7. A pattern that includes some of the Sarbanes Oxley policies

We are working on the necessary environment to use this approach, including:

- A complete catalog of security patterns including many varieties of access control models to let designers find the required solution to each security situation
- A classification of patterns according to their security concerns and architectural level. We proposed such a classification in [27].
- A tool incorporating pattern classifications and a catalog. Pattern maps would make use of pattern classifications to select which patterns to present to a designer; for example, operating system-level patterns to a system designer.
- A standard representation of security patterns. This is important for implementing tools and for a more widespread use of security patterns.

References

1. Fernandez, E. B., Larrondo-Petrie, M.M., Sorgente, T., VanHilst, M.: A methodology to develop secure systems using patterns. In: Mouratidis, H., Giorgini, P. (eds.) Integrating security and software engineering: Advances and future vision. pp. 107-126, IDEA Press (2006)
2. Mouratidis, H., Jurjens, J., Fox, J.: Towards a Comprehensive Framework for Secure Systems Development. In: Advanced Information Systems Engineering: Proc. of the 18th International Conference, CAiSE 2006. LNCS, vol. 4001, pp. 48-62. Springer, Heidelberg (2006)
3. Yoshioka, N.: A development method based on security patterns. Presentation, NII, Tokyo (2006)
4. De Capitani di Vimercati, S., Samarati, P., Jajodia, S.: Policies, models, and languages for access control. In: Databases in Networked Information Systems. Proc. of the 4th International Workshop, DNIS 2005. LNCS, vol. 3433, pp. 225-237. Springer, Heidelberg (2005)
5. De Capitani di Vimercati, S., Paraboschi, S., Samarati, P.: Access control: principles and solutions. Software - Practice and Experience. 33 5, 397-421 (2003)
6. Gollmann, D.: Computer Security. John Wiley & Sons, New York (2006)
7. Summers, R. C.: Secure Computing: Threats and Safeguards. McGraw-Hill (1997)
8. Harrison, M., Ruzzo, W., Ullman, J.: Protection in Operating Systems. Communications of the ACM, 19 8, 461-471 (1976)
9. Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman C.E.: Role-based access control models. IEEE Computer. 29 2, 38-47 (1996)
10. Sandhu, R., Bhamidipati, V., Munawer, G.: The ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security. 2 1, 105-135 (1999)
11. Thomsen, D., O'Brien, R. C., Bogle, J.: Role Based Access Control framework for network enterprises. In: Proc. of the 14th Annual Computer Security Applications Conference, pp. 50-58. IEEE Press, New York (1998)
12. Priebe, T., Fernandez, E.B., Mehlau, J.I., Pernul, G.: A pattern system for access control. In: Farkas, C., Samarati, P. (eds.) Research Directions in Data and Applications Security XVIII, Proc. of the 18th. Annual IFIP WG 11.3 Working Conference on Data and Applications Security. Springer, New York (2004)

13. Chandramouli. R.: A Framework for Multiple Authorization Types in a Healthcare Application System. In: Proc. of the 17th Annual Computer Security Applications Conference, ACSAC, pp. 137-148. IEEE Press, New York (2001)
14. Zhang, L., Ahn, G.J., Chu, B.T.: A role-based delegation framework for healthcare systems. In: Proc. of the 8th ACM Symposium on Access Control Models and Technologies, SACMAT'02, pp. 125-134 (2003)
15. Thomas, R.K.: Team-Based Access Control (TMAC): A primitive for applying role-based access controls in collaborative environments. In: Proc. of the 2nd ACM Workshop on Role-based access control, RBAC 97. pp. 13-19 (1997)
16. Barkley, J., Beznosov, K., Uppal, J.: Supporting relationships in access control using Role Based Access Control. In: Proc. of ACM Role-Based Access Control Workshop, RBAC'99, pp. 55-65 (1999)
17. Schumacher, M., Fernandez, E.B., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating security and systems engineering. John Wiley & Sons, New York (2006)
18. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns. John Wiley & Sons, New York (1996)
19. Gamma. E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston (1994)
20. Fernandez, E.B., Summers, R.C., Coleman, C.B.: An authorization model for a shared data base. In: Proc. of the 1975 ACM SIGMOD International Conference, pp. 23-31 (1975)
21. Fernandez, E.B., Pan, R.: A pattern language for security models. In: Proc. of the 8th Pattern Languages of Programs Conference, PLOP 2001, pp. 11-15 (2001) http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ebfernandezandrpan0/PLoP2001_ebfernandezandrpan0_1.pdf
22. Fernandez, E.B., Summers, R.C., Wood, C.: Database Security and Integrity (Systems Programming Series), Addison-Wesley, Reading (1981)
23. Fernandez, E.B., Pernul, G.: Patterns for Session-Based Access Control. In. Proc. of the Conference on Pattern Languages of Programs, PLoP 2006 (2006)
24. Delessy, N., Fernandez, E.B., Larrondo-Petrie, M.M.: A pattern language for identity management. In: Proceedings of the 2nd IEEE International Multiconference on Computing in the Global Information Technology, ICCGI 2007 (2007)
25. Koch, M., Parisi-Presicce, F.: Access Control Policy Specification in UML. In: Proc. of Critical Systems Development with UML, satellite workshop of UML 2002, TUM-I0208, pp. 63-78 (2002)
26. Fernandez, E.B., Yuan, X.: Semantic analysis pattern. In: Proc. of the 19th International Conference on Conceptual Modeling ER2000, pp. 183-195 (2000)
27. Fernandez, E.B., Washizaki, H., Yoshioka, N., Kubo, A., Fukazawa, Y.: Classifying security patterns. Accepted for the 10th Asia-Pacific Web Conference, APWEB'08, (2008)