Improving Scenario-Driven Role Engineering Process with Aspects

Shu Gao, Zhengfan Dai
School of Computer Science
Florida International University
Miami, FL 33199, USA
{sgao01, zdai01}@cs.fiu.edu

Huiqun Yu
Department of Computer Science
East China University of Sci & Tech
Shanghai 200237, China
yhq@ecust.edu.cn

Abstract

Role engineering for role-based access control (RBAC) is a process to define roles, permissions, constraints, and role hierarchies. The scenario-driven role engineering process provides a systematic way to elicit the RBAC components. However, the traceability between those components has not been well analyzed. As a result, it is both time-consuming and error-prone to modify the RBAC components. Therefore, a guideline to propagate changes is essential to the scenario-driven role engineering process. The importance of traceability in role engineering has been recognized but very little work has been done to date. In this paper, we propose a way to address the traceability issue based on aspect-oriented requirements engineering.

1. Introduction

The role-based access control (RBAC) is a class of access control (AC) policies in which permissions are associated with roles and users are assigned to appropriate roles [28]. A role is a job function assigned to a user within an organization that describes the authority and responsibility of the user. Usually, within an organization, roles are more stable than users and task assignments since an organization's job functions change less frequently than its employees [9]. By authorizing to roles instead of specific users, RBAC can greatly reduce the management complexity of access control. RBAC has already become a predominant access control methodology, which is used as an efficient alternative to traditional Discretionary Access Control (DAC) [15] and Mandatory Access Control (MAC) [1].

Role engineering for RBAC is a process to define roles, permissions, constraints, role hierarchies and permission-role assignment (PA) [6]. Role engineering is essentially a process of requirements engineering [17]. As Figure 1 shows, the input of role engineering process is the initial access control requirements, which are often informal, imprecise, ambiguous, incomplete and inconsistent. The product of role engineering is a concrete RBAC policy, which is an instance of abstract RBAC models defined in [28, 12]. The concrete RBAC policy is precisely defined, and subsequently, the access control system that enforces and administrates the RBAC policy can be developed.

As Clarke et al. defined in [5], traceability is the ability to determine readily how a piece of one software artifact (e.g., requirement, design, code) affects others. In this paper, we investigate the traceability issue in role engineering, and more specifically, traceability in the scenario-driven role engineering process: how changes made to part of RBAC policy will affect others. The concept of requirement level aspect is introduced, and the relationships among requirement level aspects are explicitly identified. By doing so, we are targeting at a bunch of problems that arise due to weak traceability, including: poor comprehensibility, poor evolvability, and high impact of change.

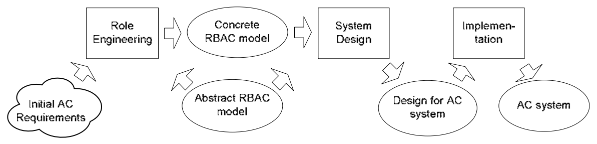

Figure 1. The RBAC system realization process

1.1. Scenario-driven role engineering process: an overview

In [20], Neumann and Strembeck propose a novel role engineering process based on scenarios. A scenario [3, 4] is a narrative of a use episode. Compared with abstract goals [19] or requirements, scenarios are easier to understand, therefore scenarios are widely used to promote communication among stakeholders, both technical and non-technical. With scenarios, it is possible to consider the strong human factor in role engineering.

Figure 2 is a high-level view of the scenario-driven role engineering process. It is a sequential workflow with iterations. The concept of scenario is the centerpiece of the process. On the one hand, permissions are derived from scenario models, i.e. a scenario usually contains actions that require several permissions. On the other, scenarios are composed to form task definitions. Several tasks can be grouped into a work profile, which is a preliminary RBAC role[1].

The identification of constraints is very complex for there are so many types of constraints. Therefore, the authors suggest using a separate sub-process for each constraint type. For example, in [21], Neumann and Strembeck describe a goal-oriented process to engineer context constraints.

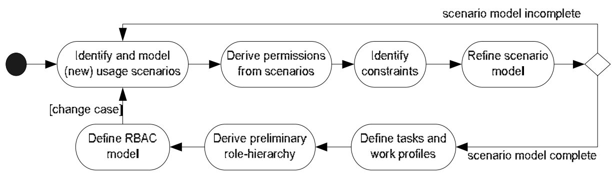
scenario model incomplete
Identify and model (new) usage scenarios
Derive permissions from scenarios
Identify constraints
Refine scenario model
[change case]
Define RBAC model
Derive preliminary role-hierarchy
Define tasks and work profiles
scenario model complete

Figure 2. A high-level view of the scenario-driven role engineering process [NS02]

1.2. Motivation

As the initial access control requirements are often imprecise, incomplete and inconsistent, the definition of RBAC components derived from the initial requirements is not stable. Therefore, during the process of role elicitation, the security engineer may make change to any of the RBAC components. Then how to propagate the change made to one RBAC component to other components is the *traceability problem*. The significance of the traceability issue for role engineering has already been identified by [20]. However, in Neumann and Strembeck's approach, the propagation of a change must start from the scenario model. And they do not analyze the relationship between RBAC components. Moreover, one limitation of the scenario-driven process is that scenarios are inherently partial [18, 20], which means scenarios can hardly cover all real-world situations. So as we mentioned at the beginning of this subsection, changes may start from any one RBAC component and must be propagated to other components. To effectively manage the propagation, the traceability links between the components must be carefully analyzed.

Applying the separation of concerns principle in requirements engineering can help acquire better traceability and ensure consistency. Aspect-oriented programming (AOP), as an effective technique to achieve separation of concerns, has already attracted more and more attention. The success of AOP naturally guides researchers to apply aspect-orientation in the early phases of software development, namely, requirement analysis and architectural design. This motivates us to investigate the prospect of using aspect-oriented requirements engineering to improve the traceability of role engineering.

The rest of this paper is structured as follows. Section 2 explains the aspect-oriented approach to improving the traceability in role engineering. In Section 3, a small example is given to show how the proposed approach works. Section 4 describes the benefits of applying aspect-orientation in role engineering as well as the future work. Section 5 is related work and Section 6 concludes the paper.

2. Improve traceability in role engineering with aspects

Role engineering is essentially a process of requirements engineering. The traceability problem in requirements engineering has been studied for a long time [14, 24]. Applying the separation of concerns principle in requirements engineering can help acquire better traceability and ensure consistency. Several methods geared toward separation of concerns have been proposed, for example: viewpoints [11], use cases and goals. However, none of them explicitly focus on crosscutting concerns [25]. At the requirement level, crosscutting concerns are concerns that cannot be encapsulated by a single viewpoint or use case. Therefore, their representations are tangled and hard to understand and maintain using the above methods. There are functional requirements and non-functional requirements. Functional requirements capture the intended use of the system. Non-functional requirements [16] are quality requirements such as security, mobility, availability, interoperability and real-time constraints. In this paper, we are interested in functional crosscutting concerns in role engineering. We do not intend to replace the scenario-driven role engineering process, which is a systematic way to generate RBAC components from the initial access control requirements. But by identifying aspects and defining their relationships, we hope after the concrete

[1] There may be many redundancies in the work profile definition, while for role definitions, redundancies are minimized.

RBAC model is defined, the change management can become more efficient.

2.1 Identification of aspects in role engineering

Crosscutting concerns are encapsulated as aspects in aspect-oriented software development (AOSD). To identify aspects in role engineering, we find crosscutting concerns, and label them as aspects.

Table 1. Relationships between aspects

	Change	Impact
Permissions	Add	Need to update scenarios
	Delete	Need to update permission constraints, PA and scenarios related to the permission deleted
Permission Constraints	Add	Need to check the consistency of PA and other permission constraints; need to update scenarios
	Delete	Need to update scenarios
Roles	Add	Need to check redundancy in role definitions; update scenarios
	Delete	Need to delete corresponding PA, adjust role constraints, and role hierarchies; need to update scenarios
Role Hierarchies	Add	Need to delete redundant PAs, check the consistency of permission constraints and role constraints; need to update scenarios
	Delete	Need to modify PAs, permission constraints and role constraints; need to update scenarios
Role Constraints	Add	Need to check the consistency of role constraints and role hierarchies; need to update scenarios
	Delete	Need to update scenarios
PA	Add	Need to check the consistency of permission constraints and role hierarchies; need to update scenarios
	Delete	Need to check the consistency of role hierarchies and update scenarios
Scenarios	Add	All other aspects
	Delete	All other aspects

A use case is a sequence of actions performed by the system to yield an observable result of value to a particular user [2]. In the scenario-driven role engineering process, a scenario is essentially equivalent to a use case [22]. The RBAC components – roles, permissions, role hierarchies, and constraints are some concerns. Can they be encapsulated into a single scenario? The answer is no. The scenario-driven role engineering process can be used as a proof. We only illustrate two cases here. First, the derivation of roles: a role comes from the definition of a work profile; a work profile consists of several tasks; each task is a collection of scenarios. So a role usually corresponds to several scenarios. Second, the derivation of permissions: while a scenario contains one or more permissions, a permission may be needed in more than one scenario. Thus, permissions cannot be captured by one scenario either. By the definition of requirement level crosscutting concern, use case, and scenario, we conclude that all RBAC components involved in role engineering are crosscutting concerns.

Finally, the aspects we identified are: Permissions, Permission Constraints, Roles, Role Hierarchies, Role constraints and Permission-role assignments (PA).

2.2 Relationships between aspects

Our purpose of identifying requirement level aspects in role engineering is to improve the traceability, since separation of concerns aids to isolate changes, minimize modification efforts and limit the impact on unrelated parts of a system [29]. After we define the aspects, the next and most important step is to explicitly identify the relationships between these aspects.

As Neumann and Strembeck point out in [20], it is a very complex task to find all possible traceability links between the aspects we have identified. However, we believe that the formation of a relatively complete knowledge base of traceability links may take a long time; but it is worthwhile to create such a knowledge base and even a preliminary one can help save time and reduce mistakes in later change management.

In this position paper, we briefly list the relationships between aspects in Table 1. There are only two basic types of changes: add and delete. Other changes such as modification can be decomposed into the deletion of the old definition, and the addition of a new one.

Most of the impacts given in the table are simply possible impacts. If no inconsistency is found, then no action is required. Please note that we list scenarios together with the other six aspects. This is because changing scenarios may affect all the other aspects, for scenarios are the sources to define all the RBAC components. Also changing any RBAC component requires updating the scenario models. The description of scenario models is an important part of the documentation for requirement analysis. Including scenarios in the table also reflects the connection between the scenario-driven process and our approach.

2.3 Change Propagation

Changing the definition of one aspect always incurs a chain of modifications. For example, if the security engineer adds a new role, he/she needs to check if there is any redundant role. If there is, the redundant role will be deleted. The deletion of a role will affect PA, role constraints, and role hierarchies. Depending on the effect of the deletion, the security engineer may need to modify these aspects. Then another set of aspects will be affected by the changes made to PA, role constraints, and role hierarchies. The scope of the impact of changes seems expanding. Fortunately, the number of aspects affected will not be greater than the total number of aspects we have identified. And the process should be feasible for automation. Figure 3 is the pseudo code for change propagation.

```
source_of_change = ∅;
change_propagation(aspectA, source_of_change)
{
    do the indicated change to aspectA;
    update scenarios;
    source_of_change = source_of_change + aspectA;
    for each aspectX in aspectA's impact list
    {
        if aspectX ∉ source_of_change
            change_propagation(aspectX, source_of_change);
    }
    return;
}
```

Figure 3. Pseudo code for change propagation

The *source_of_change* is a set used to remember the trace of the changes. Initially it is empty. *aspectA* is the aspect to be changed. The *indicated change* can be as simple as adding or deleting, and can also be a complex process such as checking for inconsistency. So the step to *do the indicated change* can be fully automated or need some human interaction. The *if* clause checks whether an aspect in the impact list of *aspectA* is actually a cause of the change made to *aspectA*. If so, this aspect will be skipped; otherwise, the change will continue to propagate. In other words, there is no cycle on the path of change propagation, which is why the number of affected aspects will not be greater than the total number of aspects.

3. A small example

The example we use is a small one based on similar stories used in many access control papers. It is a simple patient information system (SPIS). The scenario is depicted as follows.

Mr. Brown has a chronic heart disease. One day, Mr. Brown feels his heart extremely uncomfortable. He cannot drive to the hospital in that condition, so he has to call the ambulance. The ambulance comes quickly. Every ambulance can access the patient information system through wireless network (we presume the wireless communication is well protected, using techniques such as VPN). From the summary of Mr. Brown's patient record, the ambulance staffs know some necessary information, e.g. Mr. Brown's age, blood type, and his attending physician, etc. After the ambulance get to the hospital, Mr. Brown's condition is getting worse and he is sent to the ER. Unfortunately, His attending physician, Dr. Johnson is not available at this moment. In this condition, Dr. Johnson's colleague, Dr. Hope is authorized to act as Mr. Brown's attending physician, i.e. Dr. Hope can edit the current episode and read all previous episodes. After Mr. Brown's condition is safe and stable, he is taken care of by nurses. By scanning the bar code attached to Mr. Brown's wrist, a nurse can verify the name of him in the patient information system.

It is not our purpose to illustrate how to apply the scenario-driven role engineering process, so we directly give the result of the role engineering process.

Permissions (in <operation, object> pairs):
- <read_name, record>
- <read_demographic_info, SPIS>
- <read_summary, record>
- <read_episode, record>
- <modify_episode, record>

Permission constraints:

Context conditions:
- $Cond_1$: relation = attending
- $Cond_2$: record_status = open

Permissions with context constraints:
- <read_summary, record>{$Cond_1$}
- <read_episode, record>{$Cond_1$}
- <modify_episode, record>{$Cond_1$, $Cond_2$}

Roles:

Nurse, Ambulance Staff, Physician

PA:

Nurse: <read_name, record>,
  <read_demographic_info, SPIS>
Ambulance Staff: <read_name, record>,
  <read_summary, record>{$Cond_1$}
Physician: <read_name, record>,
  <read_demographic_info, SPIS>,
  <read_summary, record>{$Cond_1$},
  <read_episode, record>{$Cond_1$},
  <modify_episode, record>{$Cond_1$, $Cond_2$}.

Role hierarchy:

Nurse < Physician
Ambulance Staff < Physician

Role constraints:

None

To show how aspects-oriented requirements engineering helps improve the traceability in role engineering, we use the following change case as an example to explain how the change is propagated under the guide provided in Table 1.

Change case: Add a Patient role and a new permission constraints. Then assign a permission with the new constraint to Patient, and make the Patient role as the junior role to both Nurse and Ambulance Staff.

Although in this scenario the patient does not use the SPIS, it is reasonable to add a Patient role. The Patient role should have the permission to view the patient's own episode. And since everyone working in the hospital may also become a patient, the role hierarchy needs to be modified to reflect this. Because there is no permission constraint to test the owner of a patient record, such a constraint is added. The permission assigned to the Patient role is specified as <read_episode, record>{relation = owner}.

If the change is made in an ad hoc way, then after the required changes are made, how other RBAC components are affected is not clear, since the traceability links between those components are very complex and there is no guideline to follow. The quality of the RBAC model depends on the security engineer's experience.

If using the original scenario-driven process, to add the Patient role, the security engineer has to start with some scenario models from which the Patient role can be derived, even if he/she knows what role and permission constraint should be added. Comparing with the ad hoc way, this is more systematic, and can better assure the quality. But given the fact that most steps in the scenario-driven process cannot be automated, this is not an efficient way to propagate the changes.

With aspect-oriented approach, the security engineer can have a guideline like Table 1, in which the aspects that may be affected by the new role, permission constraint, PA, and inheritance relationship, as well as how they are affected are explicitly described. In this case, the change propagation process is depicted in Figure 4. "Ok, no change" means there is no inconsistency or redundancy found, so no further action is done to those aspects in the impact list. By default, after every change, the scenarios are updated. Though we do not have to go through the scenario-driven process again, updating the scenario models is a necessary step to maintain a good traceability. As the figure shows, the impact of change case we give is small (virtually no impact). In the worst case, a chain of changes appears. Base on our reasoning in Section 2, the length of the chain will not be greater than the total number of aspects.

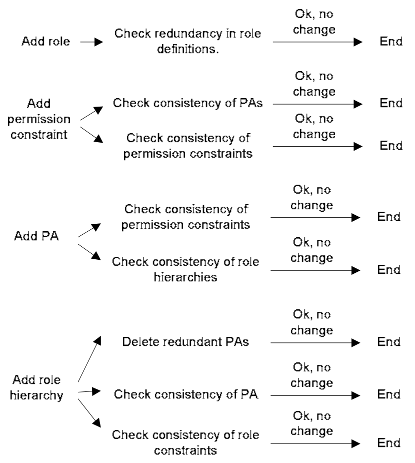

Figure 4. A change propagation process

4. Discussion

We propose this aspect-oriented approach to work with the scenario-driven role engineering process. Scenario is a good way to capture and elicit system's functions and customers' requirements. If the initial access control requirements are easy to express by a set of scenarios, then using the scenario-driven process, a concrete RBAC model can be systematically defined. Our approach works better than the scenario-driven process in managing changes, especially when the security engineer knows what changes he/she is about to make. Compared with the scenario-driven process, in which the changes must start at the scenario models, the aspect-oriented approach allows changes starting at any RBAC component and propagates them directly. While the scenario-driven process depends on human interaction significantly, most steps for propagating changes among aspects can be automated.

The significance of the relationships between aspects is that they are guidelines to analyze RBAC policies. As a result, they can help minimize both the modification effort and the probability of mistakes. Also, they are necessary to implement role engineering tools.

As Figure 1 shows, role engineering is a part of the RBAC system implementation process. To build a flexible and extensible RBAC system that supports different RBAC models, we have introduced aspect-oriented design approach in [13] and seen the benefits. By using aspects early in the requirement analysis phase, we expect to acquire a better alignment of requirement, design, and code. The requirement level aspects we have defined in this paper can be mapped into the design level aspects defined in [13]. This improves the traceability between requirements and code.

The aspect-oriented approach for role engineering is not mature yet. The scope of the aspects should cover more crosscutting concerns. The granularity of aspect definitions need to be refined. For example, there are several types of constraints, most of which should be defined into a different aspect. The relationships between aspects are a key part which needs significant improvement. What we list in this position paper is very preliminary. It should be incremented, refined, and formalized.

5. Related work

The traceability problem for role engineering has not received enough attention despite its importance. Among multiple papers we reviewed on role engineering [6, 10, 7, 26, 8, 20, 27], the traceability issue is explicitly acknowledged only in [20]. And in papers that do attempt to address the traceability problem, approaches are often primitive and inefficient.

In [20], the traceability links between RBAC components are established based on the scenario-driven role engineering process. The process limits that the changes must first be made to scenarios, and only then they can be propagated following the role engineering process. Since most steps in this process need human interaction, it is not an efficient way to manage changes.

Role engineering is a special application of requirements engineering. In the more general topic of the latter, researchers have already employed aspects as well as other notions such as features [23] and subjects [5], to encapsulate requirement level crosscutting concerns. Two representative works are [25] and [30].

In [25], Rashid el al. propose an AORE model to modularize and compose crosscutting requirements. In addition, they provide a concrete model with viewpoints and XML. Their approach offers a good way to identify and manage conflicts that arise due to tangled representations.

In [30], Sutton and Rouvellou introduce Cosmos, a general-purpose concern-space modeling schema. Cosmos supports concerns that span the life cycle of software development and are independent of particular development tools or artifacts. The authors classify concerns into logical and physical ones and identify four categories of relationships to support traceability.

A common problem in both [25] and [30] is that while the composition rules or relationships they give are quite comprehensive, the semantics of those rules or relationships are far from being formal. This makes these rules or relationships difficult to analyze and reuse.

6. Conclusion

Access control requirements are subject to a continuous change process. Changes in one RBAC model components often incur a chain of changes in other components. Consequently, the traceability issue in role engineering for RBAC becomes very important. We show that RBAC model components – roles, permissions, role hierarchies, constraints, and permission-role assignments are requirement level crosscutting concerns which cannot be encapsulated by a single scenario. As a result, their representations are tangled and hard to maintain. To make a change to the RBAC model, one has to start from the scenario model and go through the process, which is of low efficiency. In this paper, based on the scenario-driven role engineering process, we introduce an aspect-oriented approach to improve the traceability. RBAC components are identified as requirement level aspects. Furthermore, the relationships between these aspects are preliminarily defined. This way, the security engineers have a guideline to follow and can propagate changes more efficiently.

The aspect-oriented approach for role engineering has a good perspective. Besides improving the traceability of scenario-driven role engineering process, this approach can also help analyze the consistency of the RBAC policies. It is a key step toward an aspect-oriented framework for RBAC system realization.

7. References

[1] D. E. Bell and L. J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. M74-244, Mitre Corporation, Bedford, Massachusetts, 1975.

[2] G. Booch, J. Rumbaugh and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley Longman, Inc, 1999.

[3] J. M. Carroll (ed.). Scenario-Based Design: Envisioning Work and Technology in System Development. John Wiley & Sons, 1995.

[4] J. M. Carroll. Five Reasons for Scenario-Based Design. Proc. of the 32nd Hawaii International Conference on System Sciences, 1999.

[5] S. Clarke, W. Harrison, H. Ossher, and P. Tarr. Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code. Proc. of OOPSLA, 1999.

[6] E. J. Coyne. Role Engineering. Proc. of the ACM Workshop on Role-Based Access Control, 1996.

[7] P. Epstein and R. Sandhu. Engineering of Role/Permission Assignments. Proc. of the ACM Workshop on Role-Based Access Control, 1999.

[8] P. Epstein and R. Sandhu. Engineering of Role/Permission Assignments. Proc. of the 17th Annual Computer Security Applications Conference (ACSAC), December 2001.

[9] D. F. Ferraiolo, D. M. Gilbert and N. Lynch. An examination of federal and commercial access control policy needs. In NIST-NCSC National Computer Security Conference, pages 107-116, Baltimore, MD, September 20-23, 1993.

[10] E. B. Fernandez and J. C. Hawkins. Determining Role Rights from Use Cases. Proc. of the ACM Workshop on Role-Based Access Control, 1997.

[11] A. Finkelstein and I. Sommerville. The Viewpoints FAQ. BCS/IEE Software Engineering Journal, 11(1), 1996.

[12] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security, vol. 4, pp. 224-274, 2001.

[13] S. Gao, Y. Deng, H. Yu, X. He, K. Beznosov, and K. Cooper. Applying Aspect-Orientation in Designing Security Systems: A Case Study. Proc. of International Conference of Software Engineering and Knowledge Engineering, 2004.

[14] O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. Proc. of the IEEE International Conference on Requirements Engineering (ICRE), 1994.

[15] M. A. Harrison, W. L. Ruzzo and J. D. Ullman. Protection in Operating Systems. Communications of ACM. Volume 19. No. 8. August 1976.

[16] S. E. Keller, L. G. Kahn and R. B. Panara. Specifying Software Quality Requirements with Metrics. Tutorial: System and Software Requirements Engineering, R. H. Thayer and M. Dorfman, Eds., IEEE Computer Society Press, 1990, 145-163.

[17] G. Kotonya and I. Sommerville. Requirements Engineering – Processes and Techniques. John Wiley & Sons, 1998.

[18] A. Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. Proc. of International Conference on Software Engineering, 2000.

[19] A. Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. Proc. of the 5th IEEE International Symposium on Requirements Engineering (RE), August 2001.

[20] G. Neumann and M. Strembeck. A Scenario-Driven Role Engineering Process for functional RBAC Roles. Proc. of the 7th ACM Symposium on Access Control Models and Technologies, 2002.

[21] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. Proc. of the 8th ACM Symposium on Access Control Models and Technologies, 2003.

[22] OMG. OMG Unified Modeling Language Specification. March 2003, Version 1.5.

[23] J. Pang and L. Blair. Refining Feature Driven Development – a Methodology for Early Aspects. In Early Aspects Workshop in conjunction with the 3rd International conference on Aspect-Oriented Software Development, 2004.

[24] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, 27(1), January 2001.

[25] A. Rashid, A. Moreira, and J. Araujo. Modularisation and Composition of Aspectual Requirements. Proc. of International Conference on Aspect-Oriented Software Development (AOSD), pp.11-20, 2003.

[26] H. Roeckle, G. Schimpf, and R. Weidinger. Process-Oriented Approach for Role-finding to Implement Role-Based Security Administration in a Large Industrial Organization. Proc. of the ACM Workshop on Role-Based Access Control, 2000.

[27] D. Shin, G-J. Ahn, S. Cho, and S. Jin. On Modeling system-Centric Information for Role Engineering. Proc. of the 8th ACM Symposium on Access Control Models and Technologies, 2003.

[28] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-base access control models. IEEE Computer, vol. 29, no.2, 1996, 38-47.

[29] S. M. Sutton Jr. and I. Rouvellou. Concern Space Modeling in Cosmos. OOPSLA 2001 Poster Session, October, 2001, Tampa, Florida.

[30] S. M. Sutton Jr. and I. Rouvellou. Modeling of Software Concerns in Cosmos. Proc. of International Conference on Aspect-Oriented Software Development (AOSD), 2002.