# Representing Access Control Policies in Use Cases

Khaled Alghathbar
Center of Excellence in Information Assurance, College of Computer and Information Sciences, King Saud University, Saudi Arabia

**Abstract:** *Security requirements of a software product need to receive attention throughout its development lifecycle. This paper proposes the required notation and format to represent security requirements, especially access control policies in use case diagram and use case description. Such enhancements offer simple representation for positive and negative authorization, grouping sensitive use cases that form a critical business task, separation of duties – both static and dynamic, least privilege, inheritance of authorizations, and security state or label for data inputted, stored or outputted. Validating information flow requirements at an early stage prevents costly fixes that are mandated during later stages of the development life cycle.*

**Keywords:** *Access control policies, security engineering, use cases, misuse.*

*Received December 14, 2009; accepted May 21, 2010*

## 1. Introduction

Security requirements of a software product need to receive attention throughout its development lifecycle. Because the security requirements specified at early stages of the life cycle affect later stages and are likely to feature in the eventual product, it is important to specify them precisely and unambiguously with sufficient details. Security policies have not been integrated with mainstream system requirements, perhaps because they were considered as a non-functional aspect of software systems. As claimed by Nuseibeh and Easterbrook, "Non-functional requirements (also known as quality requirements) are generally more difficult to express in a measurable way, making them more difficult to analyze" [17]. Therefore, we propose several extensions to Unified Modelling Language (UML) [18] to represent access control policies. Our proposal is consistent with Devanbu and Stubblebine's [8] challenge to adopt extensions to standards such as UML for modelling security-related features. Our proposal currently covers the requirement specification and analysis phases, and our ongoing work addresses their adaptation and effect on later phases.

In UML, requirements are specified with use cases at the beginning of the life cycle. Use cases specify actors and their intended usage of the envisioned system. Such usage - usually, but not always - is specified in terms of the interactions between the actors and the system, thereby specifying the behavioral requirements of the proposed software. Fowler and Scott say that, "a use case is a set of scenarios tied together by a common user goal" [12]. Use cases are written in an informal natural language. Thus, different people may write varying degrees of details for the same use case.

Currently, a use case is a textual description with:

1. Actors and/or their roles.
2. Preconditions and post conditions.
3. Normal scenarios with sequence of actions by the actors and/or the system.
4. Abnormal or exceptional scenarios.

In contrast, a use case diagram visualizes actors and their relationships with scenarios [5, 13]. As we shall demonstrate during the course of this paper, use cases are not sufficient to model the details of access control policies. Consequently, we propose that use cases need to be enriched with something analogous to (soon to be discussed) operation schemas that we refer to as access control policy schemas.

"Operation schemas introduced by Sendall and Strohmeier [22] enriches use cases by introducing conceptual operations and specifying their properties using Object Constraints Language (OCL) syntax [26]. An operation schema specifies operations that apply to the whole system to be taken as one entity. One of the advantages of operation schemas is that they can be directly mapped to collaboration diagrams that are used later in the analysis and design phases. It is our position that high-level access control policies should be applied at this level of detail."

Although operation schemas are precise, they do not specify system security. Therefore, we extended the operation schemas to cover access control, and we refer to the extended schemas as access control policies schemas. Introducing access control schema as a separate syntactic entity has several advantages. Firstly, it isolates access control policies from other functional requirements that are usually elaborated in operation schemas. Secondly, this separation facilitates several access control policies to one use case, thereby modularizing the design.

Unlike most functional requirements, access control policies express many constraints that are to be enforced by the system. For example, an employee - no matter the rank in the organization must not be able to approve his/her own salary increases. We show how to specify and enforce such requirements using access control schemas. In addition, use cases have dependencies between them. For example a check cannot be approved before it is written. We show how such constraints can be specified and resolved using access control schemas.

There is a need for negative authorization as there is a need for positive authorization. In particular, with the presence of subject hierarchy, the need for explicit negative authorization is greater, because subjects do not have explicit authorizations only but also it may have implicit authorizations as well from the inheritance of junior subject's permissions. Therefore, negative authorizations are used to block some positive authorizations that have been granted to a subject. With the introduction of negative authorization, there is also a need to manage any conflict between authorizations (positive and negative).

Work has been done by Sindre and Opdahl [24, 25] to enrich use case diagram and its description with what is called as misuse cases. Misuse cases-which have been used in their industry [4] were introduced to enhance the use case diagram to represent threats or abuses scenarios that users do not want to happen and must be prevented or mitigated. Along with misuse cases which represent the scenario, mis-actor also was introduced to represent special kind of actor who invokes the misuse cases [24, 25]. However, misuse cases or other work cannot represent all security threats or requirements such as access control and flow control policies which must be integrated into the original use cases as those policies are related to the authorized actor not just the mis-actor.

This paper expands and enhances the work done in [3] by improving the representation and capturing of security requirement especially access control policies and expanding the work to capture new access control policies in a simpler way. The representation of access control policies in this paper allows analysts to capture and represent the following access control polices:

- Positive authorization.
- Negative authorization.
- Grouping sensitive use cases that form a critical business task.
- Static separation of duties.
- Dynamic separation of duties.
- Least privilege authorizations.
- Inheritance of authorizations.
- Security state of data inputted, stored and outputted.

Section 3 shows how those policies are represented use case description while section 5 shows how those policies are represented in use case diagram according to our proposed addition.

The remainder of this paper is organized as follows. Section 2 explains an example that will be used throughout the paper. Section 3 shows the proposed enhancement of use case description. Section 4, shows the application constraint representation of separation of duties policies. Section 5 demonstrates the proposed enhancement of the use case diagram. Section 6 discuses the proposed enhancements and how our work relates to other contributions and section 7 concludes the paper.

## 2. Running Example

The running example describes a purchasing process where a set of tasks assigned to authorized roles as shown in Figure 1. Role-Based Access Control (RBAC) [21] is the access control model for this example. The set of access control policies applicable to this example are as follows:

1. Use cases such as record invoice arrival, verify invoice validity, authorize payment and write a check are to be applied in the specified order.
2. Each use case should be executed by an actor playing an authorized role(s) as shown in Figure 1. For example, write a check use case should be invoked by (authorized to) clerk role. In addition, the role hierarchy implicitly authorizes a specialized role to inherit permissions. For example, according to Figure 2, supervisor role inherits purchasing officer's permissions and purchasing officer inherits clerk's permissions.
3. Supervisor must not execute the write a check use case.
4. No user should perform more than one use case on each object. This one type of Dynamic Separation of Duty (DSOD) policy. For example, a user should not record and verify the same invoice. This policy is claimed to prevent fraud and errors [6].
5. If the invoice's total amount exceeds one million, then two different supervisors must authorize the invoice.

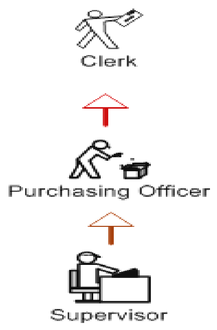

Figure 1. Use case diagram for purchasing payment example.

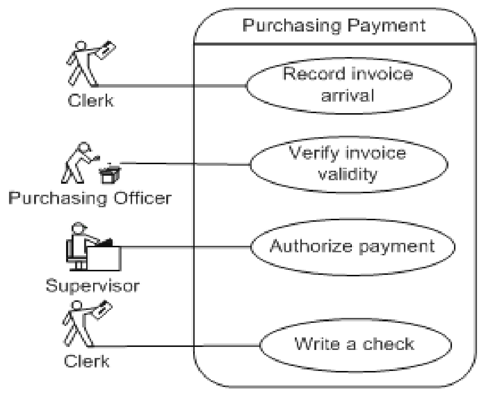

Figure 2. The role hierarchy for the running example.

## 3. Writing Access Control Policies in the Use Case Description

As we discussed in section 1, operation schemas do not cover access control policies. Therefore, we show access control policy schema to specify them. Also, we proposed several new attributes to the use case description/ template that capture the access control policies and labelling of the data.

Figure 3 shows a combined format of a use case description from Kulak and Cockburn [7] and Guiney [15].

*Use Case: the use case name.*
*Primary Actor: user who invoke this use case.*
*Iteration: denoting how refined the description is.*
*Level: level of abstraction this use case in.*
*Stakeholders and Interest: Stakeholders*
*Summary: short textual description of the action.*
*Basic Course of events: the successful action of the use cases in the form of steps.*
*Alternative Paths: less common path than the basic course.*
*Exception Paths: Path in case of errors*
*Extension Points: steps in the use case from where the extending use case diverge.*
*Triggers: triggers that initiate this use case.*
*Assumptions: valid conditions for normal execution, but not necessary.*
*Preconditions: conditions that must be met before the execution of the use case.*
*Postconditions: conditions that must be met before the execution of the use case.*
*Related Business rules: rules for during operational stage.*
*Author: Author*
*Date: Date*

Figure 3. Use case description/ template.

Figure 4 shows our proposed enhancement to the use case description that captures the access control policies and security. Figure 5 refers to the authorize a payment use case of Figure 1. It allow the analyst to specify the security measures, the required state of data flowing and which use case proceeds this use case to capture the order of use cases and also it allows an analyst to specify who is the actor that must have a positive permission to invoke this use case.

*Object: the object of the use case.*
*Business Task Group: which group of use case that form a critical business task*
*Security Measure: type of necessary security measure.*
*Security Label (Input): specifying the required security label and condition for the inputted data, i.e. Hashed, Encrypted, Plaintext.*
*Security Label (Stored): specifying the required security label and condition for the stored data, i.e. Hashed, Encrypted, Plaintext.*
*Security Label (Output): specifying the required security label and condition for the outputted data, i.e. Hashed, Encrypted, Plaintext.*
*Proceeds: which use case it proceeds*
*Least Privilege for: shows which actors must be permitted for this use case as a least privilege for him/her.*
*Declares: constants, variables, objects and data types used in the pre and post conditions.*
*Authorized (user, group, and role): a list of users, groups or roles that are authorized to access this operation on this object.*
*Denied (user, group, and role): a list of users, groups or roles that are denied to access to this operation on this object.*
*Integrity Constraints (Pre): specify all integrity constraints that must be satisfied before executing the operation written in OCL.*
*Integrity Constraints (Post): specify all integrity constraints that must be satisfied after the operation is executed. It is written in OCL.*

Figure 4. Proposed use case description attributes.

*Use Case: Authorize Payment*
*Object: Invoice*
*Business Task Group: processing a payment*
*Security Measure: 1- Authentication must be at least two factor authentication*
*2- Must be logged*
*3- Authorizing more than one million dollars must be authorized by the*
*approval of two supervisors*
*4- User cannot authorize a payment that he/she verify or record*
*the arrival of invoice*
*Security Label (Input): encrypted*
*Security Label (stored): encrypted*
*Proceeds: "write a check" use case*
*Least Privilege for: Supervisor*
*Description: Actor authorizes the payment after it has been verified. If the amount exceeds one million dollar then the authorization is partial until a different supervisor completes it.*
*Declares:*
*UserWhoDidPreviousOperations: Set(History_Log) ::= History_Log→ select (User= CurrentUser AND (Operation="Record_Invoice_Arrival" OR Operation="Verify_Invoice_Validity")AND Object= CurrentObject);*
*--it will return a record or more if the current user has done one of the previous use case.*
*Authorized (User, Group, Role): Supervisor--Role*
*Denied (User, Group, Role): none*
*Integrity Constraints (Pre):*
*Invoice.verified="true";*
*Invoice.TotalAmount<=1000000 implies Invoice.authorized= "false";*
*Invoice.TotalAmount>1000000 implies*
*(Invoice.partialAuthorized= "false" OR Invoice.authorized= "false")*
*UserWhoDidPreviousOperations → isEmpty; -- The current user did not do other operation on the current invoice(Dynamic Separation Of Duty)*
*Integrity Constraints (Post):*
*If (invoice.TotalAmount>1000000 AND invoice.partialAuthorized@pre="false") then --the*
*invoice has not been partially authorized by different Supervisor before.*
*Invoice.partialAuthorized="true";*
*else*
*invoice.authorized= "true";*
*Endif;*

Figure 5. The improved use case description for the authorize payment use case.

What we mean by least privilege here is the minimum number of authorized use cases necessary for an actor to accomplish his/her work, least privilege principle differs from simple authorization in way that least privilege is a superset of authorization rather than simple one authorization, denying actor from one of the use cases that are part of a least privilege chain of use cases for that actor will jeopardize the whole least privilege principle, while denying an authorized actor to an authorized use case may not break a bigger picture. The least privilege for the verify invoice validly use case should be purchasing officer and supervisor, but this does not imply an explicit authorization, it is just a condition that must be considered when compiling access control policies to make sure that all actors have access to their use case and their permissions do not conflict with other permissions. The pre-condition of the schema in Figure 5, has four constraints:

1. The invoice is already verified.
2. If the invoice's total amount is less or equal to one million, then the invoice must not be authorized yet.
3. If the invoice's total amount exceeds one million, then either the invoice is not yet partially authorized or partially but not fully authorized.
4. The current user did not participate in any prerequisite operation on the same invoice. Conversely, the post condition ensures the correctness of operations with respect to the access control constraints.

## 4. Constraints

Authorizations in the form of authorized or denied clauses in the access control schema do not capture all access control constraints. Therefore, there is a need to properly express application constraints such as dynamic separation of duty. Next, we will provide some access control constraints in commercial systems, and we will consider several known versions of Separation of Duty (SOD) policies. We show how to write SOD policies as an OCL constraint in the integrity constraint clause of the access control policy schema. Figure 6 illustrates the relationship between objects that are used to specify integrity constraints.

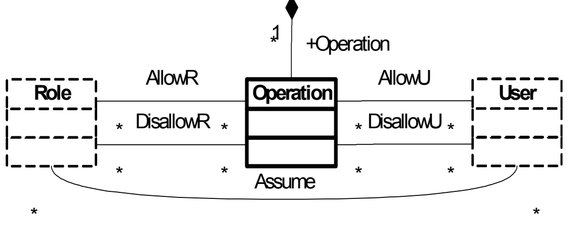

Figure 6. The access control model.

### 4.1. Static Separation of Duty Principles

Static SOD principles prevent subjects (role or user) from gaining permissions to execute conflicting operations. There are many kinds of static SOD policies and they are listed below:

1. *Mutually Exclusive Roles*: A user shall not assume two conflicting roles. For example, a user must not assume both Purchasing Officer and Accounts Payable Manager roles. This policy can be ensured if no user is enrolled in two mutually exclusive roles, say $Role_A$ and $Role_B$ and can be specified in OCL as follows:
   (Role→ select(name= "$Role_A$")).user→
   intersection(Role→select(name="$Role_B$").user)→si ze=0
2. *Business Task*: A user must not execute a specified business task that comprises a set of operations. For example, user U must not be authorized to perform Record, Verify and Authorize on the same object and this can be specified as follows:
   User.$Allow_U$→
   select(Operation=$Operation_1$ OR
   Operation=$Operation_n$) → size<n
   Where $n$ is the number of operations to perform a critical task.
3. *Mutually Exclusive Operations*: Mutually exclusive operations must not be included in one role, i.e., writing and signing a check must not be allowed to the Manager role.
   $Operation_A$.$Allow_R$→intersection
   ($Operation_B$.$Allow_R$)→size=0

### 4.2. Dynamic Separation of Duty Principles

Dynamic Separation of Duty (DSOD) allows user to assume two conflicting roles but not to use permissions assigned to both roles on the same object.

There are several types of this policy discussed in [23], of which we will show some. One DSOD constraint is to restrict user from performing Record, Verify and Authorize use cases on the same object. In order to specify this policy, a history of already granted authorizations must exist. For this purpose, we propose a formal syntactic object History_Log to maintain a Table of (user, role, operation, object and time), for deeper discussion of History_Log where refer reader to [1].

Using the History Log, we specify some DSOD principles in use.

- *Dynamic Separation of Duty*: This version says that a user cannot perform more than $n$ operation on the same object, stated as a precondition of an operation:
  History_Log→ select (User= CurrentUser AND
  (Operation=$Operation_1$ OR Operation=$Operation_2$
  OR Operation=$Operation_{n-1}$) AND Object=
  CurrentObject)→ size<n-1

### 4.3. Other Access Control Constraints

- *Role prerequisites*: A user must be enrolled in a particular role before assuming another role. This can be stated as a postcondition of the role assignment where $\text{Role}_B$ is the prerequisite role as follows:
  User.Role→ includes($\text{Role}_A$) implies User.Role→ includes($\text{Role}_B$)
- *Permission Prerequisites*: A role must be authorized to execute a particular operation before granting that role with another operation. This constraint can be specified as a postcondition of permission assignment where $\text{Operation}_B$ is the prerequisite permission. For example, the *Supervisor* role can not assume authorize a payment unless this role already has a permission to read the invoice's data.
  Role.Operation→ includes($\text{Operation}_A$) implies Role.Operation→ includes($\text{Operation}_B$)
- *Cardinality Constraints*: This constraint specifies a maximum and/or a minimum number of operations that can be executed by a user or a role. This policy may be applied to the number of users for each role or to the number of permissions for a specific role. For example, Supervisor role must have at most one user. This constraint can be specified as follows:
  (Role→select(name=RoleName)).User→ size <sign> n, where <sign> is one of the following (<,>,<=,>=,<>,=) and *n* is the limit.
  (Role→select (name=RoleName)).Operation→ size <sign> n, where <sign> is one of the following (<,>,<=,>=,<>,=) and *n* is the limit.

## 5. Drawing The Refined Use Case Diagram

Although use case diagrams visually represent the behavioral requirements of a proposed software system, they are not sufficient to represent existing access control policies. At best, a use case diagram shows some access control by stating the roles that actors are permitted to invoke.

Thus, having visual representations of access control policies is very much in accordance with the objectives of UML. We propose a refined use case diagram Figure 7 for the our running example. The refined use case diagram represents all possible access control policies (positive, negative, explicit, implicit, limited and integrity constraints), which provides clear visual access control policies. The proposed refined use case diagram has many desirable features as follows:

- A new stereotype <<Deny>> represents a deny-negative- permission for specific actor to specific use case.
- A new stereotype <<limited>> represents a limited positive authorization, in another word, this type of authorization is not obsolete, it has limited authorization according to special condition like the requirement of multiple actors action to run this authorization. We use it in Figure 7 to denote a limited permission to the supervisor to the Authorize payment use case because in the running example states that another Supervisor is required to complete a payment authorization over one million. It is enough to state this tag and leave the details of the limitation in the use case description to reduce complexity.
- The new refined use case diagram adopts a relationship, which is introduced by the Open Modeling Language (OML), called Precedes [11]. The relationship is used to specify dependencies and order of invocation among use cases.
- Inheritance of permission requirement between actors is captured by the large-head arrow, and reflected in the explicit authorization arrow that points to the use case that he/she has an explicit authorization or implicit authorization derived from inheritance. For example, because purchasing officer is a specialized actor of clerk thus purchasing officer inherits clerk permissions, as result, purchasing officer has permission to both verify invoice validly (explicit permission) and record invoice arrival (implicit permission from inherence) use cases and it is represented by pointing to a inner box (titled Business Task 3) that contains the two use cases.
- Having an inner box does not reduce the number of arrows from actors to use case only, but also to allow analyst to represent one of the access control polices which is separation of duties. The running example restricts any actor to invoke at most one use case out of all, for a specific object, even if that actor has a permission to invoke all. In another word, although Supervisor has permission explicit or implicit to invoke the first three use cases, he/she must not be allowed to invoke all of them for the same invoice to reduce the fraud, but he/she can invoke any of the use cases for different invoices and payments. We introduced a representation of this policies in the form of N:M, where N denote the maximum number of use case the actor can invoke out of M which is the total number of use cases he/she has permission on.
- There is no guarantee that this notation will provide completeness by just representing those notations in one use case diagram. Another work called AuthUML [2] focused on introducing a logic based language that checks the compatibility and completeness of the access control policies especially when tens or hundreds of access control policies are embossed in multiple use cases in large software systems.

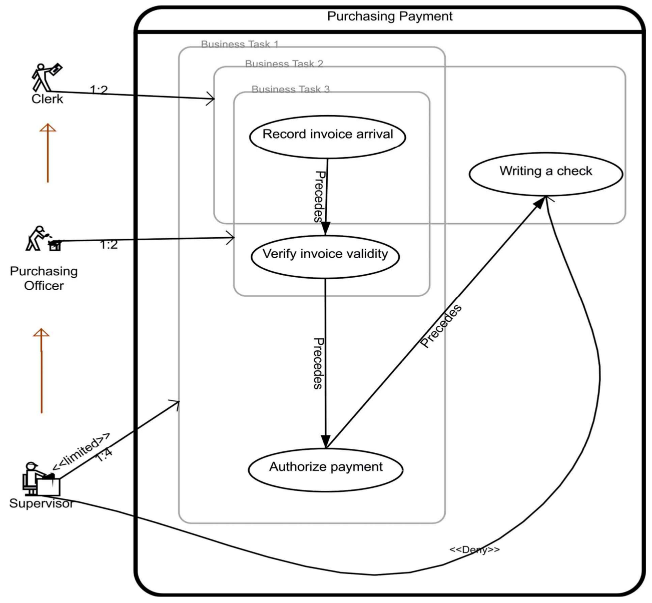

Figure 7. The refined use case diagram.

## 6. Discussion and Related Work

It is already stated that security requirements must be considered in early stages to integrate well with other requirements in the analysis, design, coding and testing. The use case is part of the tools that capture software requirements and it meant to be simple enough to allow users to be involved in the analysis phase. However, what we have introduced in this paper and what others introduced in their work might be observed as complicating the use case which contradicts with the purpose of the use case.

That is true, but security requirements are considered design-level and sometimes a programming-level [25] yet, it must be captured in the analysis phase where user or customer is involved. Thus, we do not consider what we have introduced a complexity add-on, but rather a mandatory representation for a mandatory requirements (security requirements) if software developer cares about thinking about security requirements in the early stage of the software development life cycle.

To come up with a recommendation that comply with both sides of the debate, we could recommend an iterative process of embossing access and flow control policies into the use case, starting from simple use case as one stage then adding security requirements in another subsequent stage. It is something like summary use case and detailed use case. However, it is a mandatory process to capture user requirements instead of leaving it vague for designer or programmer or even tester! Fernandez and Hawkins [9] proposed to extend use cases with rights. The extension is by means of a stereotype that states the access constraints. In addition, they propose an approach to generate rights for roles. Their work and ours have some common objectives, but the following differences exist:

1. They extend the use case by embedding security-related requirements in the use case itself, cluttering the use case. Our proposed schemas detaches access control policies from use cases and writes constraints in OCL.
2. Their work does not address complications arising from hierarchies and how to resolve access control conflicts.

Sendall and Strohmeier [22] introduced the concept of operation schemas to describe the effect of system operation and its functionality. Operation schemas supplement the use cases and is written in OCL. In addition, there is a one-to-one mapping between operation schemas and collaboration diagrams. We extended the operation schemas to represent the access control policy in what we have called access control policy schemas that focuses on access control and integrity constraints that are related to access control.

Fernandez-Medina *et al.* [10], propose an extension to the use case and Class models of UML. The extensions of use case diagram which they introduced were stereotypes: <<safe-UC>> and <<accredited - actor>> as an indication of a secure use case and authorized actor. As shown in this paper, stereotypes are insufficient to specify access control policies of a system. The extension does not address the type of authorization that is granted to the accredited actor, nor the integrity constraints associated with such authorizations.

Brose *et al.* [14], extended UML to support the automatic generation of access control policies in order to conFigure a CORBA-based infrastructure. They specify permissions and prohibitions on accessing system's objects (such as read, write, execute, etc.,) explicitly by writing notes that are attached to actors in the use case diagrams. The use case extension which they proposed is similar to the one we proposed in addressing role hierarchies and negative authorizations. However, their work is based on specification of static access control policies. It is not flexible enough to specify dynamic access control policies such as DSOD, nor can it enforce a flow control requirement such as the order of operations in a workflow system. We also based our access control specification on OCL rather than a natural language. Although, their work considers role hierarchies, no propagation or conflict resolution policies have been addressed for the inherited authorizations.

Alghathbar *et al.* introduced in [1] an extension to the UML metamodel with an access control policy constraint specification and enforcement module, business tasks and history log for method calls. The extension shows how access control requirements of an application can be modeled in the design phase. In contrast, this paper focuses on representing of access control policies on the requirement phase. In relation to this work, Alghathbar introduced in [2] AuthUML, which is a logic programming based framework that analyzes static access control requirements in the requirements phase of the life cycle to produce a consistent, complete and conflict-free access control requirements.

Work has been done by Sindre and Opdahl [24, 25] to enrich use case diagram and its description with what is called as misuse use cases. Misuse use cases - which have been used in the industry [4], introduced to enhance the use case diagram to represent threats or abuses scenarios that user do not want to happen and must be prevented or mitigated. Along with misuse which represent the scenario, mis-actor also was introduced to represent special kind of actor who invoke the misuse cases [24, 25].

Okubo and Tanaka [19] extended the misuse cases model and presents an approach for identifying security aspects and point cuts in a requirement analysis stage. Matulevicicius *et al*. [16] aligned misuse cases with security risk management. Also, Pauli and Xu [20] introduced an approach to the architectural design and analysis of secure software systems based on the system requirements elicited in the form of use cases and misuse cases.

However, misuse cases or other work introduced cannot represent all security threats or concern such as access control and flow control policies which must be integrated into the original use cases as those policies are related to the authorized actor not just the mis-actor. Thus, we introduced in this paper a refined work of [3] after considering more access control policies and controls.

## 7. Conclusions

To achieve better security, security requirements should be addressed in all phases of the development life cycle. This paper introduced a new notation and format to capture and represent more access control policies in the use case diagram and use case description. Those policies are positive and negative authorization; grouping sensitive use cases that form a critical business task; separation of duties – both static and dynamic; least privilege; inheritance of authorizations; and security state or label for data inputted, stored or outputted. This proposed work falls in the effort to provide more tools and notations to think and embed security requirements in the early stage of the life cycle. Other efforts were introduced elsewhere that complement this work to capture more security requirements other than access control policies.

There are need and room to compile and standardize those efforts. Also, there is room to go further in thinking of representing security requirements not just in use cases but also in other UML tools and in integrating UML tools into the programming phase.

## References

[1] Alghathbar K. and Wijesekera D., "Modeling Dynamic Role-Based Access Constraints using UML," *in Proceedings of the International Conference on Software Engineering Research and Applications*, USA, pp. 1-15, 2003.

[2] Alghathbar K. and Wijesekera D., "Validating the Enforcement of Access Control Policies and Separation of Duty Principle in Requirement Engineering," *Journal of Information and Software Technology*, vol. 49, no. 2, pp. 142-157, 2007.

[3] Alghathbar K. and Wijesekera D., "Consistent and Complete Access Control Policies in Use Cases," *in Proceedings of 6th International Conference on Unified Modeling Language*, CA, pp. 44-49, 2003.

[4] Alexander I., "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software*, vol. 20, no. 1, pp. 58-66, 2003.

[5] Booch G. and Rumbaugh J., *The Unified Modeling Language User Guide*, Addison-Wesley, UK, 1999.

[6] Clark D. and Wilsonv D., "A Comparison of Commercial and Military Computer Security Policies," *in Proceedings of IEEE Symposium on Security and Privacy*, CA, pp. 184-193, 1987.

[7] Cockburn A., *Writing Effective use Cases*, Addison-Wesley, 2001.

[8] Devanbu P. and Stubblebine S., "Software Engineering for Security: A Roadmap," *in Proceedings of the Conference on the Future of Software Engineering*, USA, pp. 227-239, 2000.

[9] Fernandez E. and Hawkins J., "Determining Role Rights from Use Cases," *in Proceedings of 2nd ACM Workshop on Role-Based Access Control*, USA, pp. 121-125, 1997.

[10] Fernandez-Medina E., Martinez A., Medina C., and Piattini M., "Integrating Multilevel Security in the Database Design Process," *in Proceedings of the 6th Biennial World Conference on the Integrated Design and Process Technology*, CA, pp. 255-259, 2002.

[11] Firesmith S., Henderson-Sellers B., and Graham I., *OPEN Modeling Language Reference Manual*, SIGS Books, USA, 1997.

[12] Fowler M. and Scott K., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, UK, 2003.

[13] Jacobson I., *Object-Oriented Software Engineering: A Use Case Driven Approval*, Addison-Wesley, 1992.

[14] Koch M., Parisi-Presicce A., and Pauls K., "Access Control Specification in UML Integrating Security and Software Engineering: Advances and Future Vision IDEA Group Inc," *Technical Report*, 2006.

[15] Kulak D. and Guiney E., *Use Cases: Requirements in Context*, ACM Press, 2000.

[16] Matulevicius R., Mayer N., and Heymans P., "Alignment of Misuse Cases with Security Risk Management," *in Proceedings of the 3rd International Conference on Availability, Reliability and Security*, Spain, pp. 268-372, 2008.

[17] Nuseibeh B. and Easterbrook S., *Requirements Engineering: A Roadmap in A Finkelstein*, ACM Press, 2000.

[18] Object Management Group, *OMG Unified Modeling Language Specification*, available at: http://www.uml.org/, last visited 2009.

[19] Okubo T. and Tanaka H., "Identifying Security Aspects in Early Development Stages," *in Proceedings of the 2008 3rd International Conference on Availability Reliability and Security*, Spain, pp. 742-748, 2008.

[20] Pauli J. and Xu D., "Misuse Case-Based Design and Analysis of Secure Software Architecture," *in Proceedings of the International Conference on Information Technology: Coding and Computing*, USA, pp. 522-526, 2005.

[21] Sandhu S., Coyne J., Feinstein L., and Youman E., "Role-Based Access Control Models," *Journal of IEEE Computer*, vol. 29, no. 2, pp. 3-7, 1996.

[22] Sendall S. and Strohmeier A., "Using OCL and UML to Specify System Behavior," *in Proceedings of Object Modeling with the OCL*, Berlin, pp. 250-279, 2002.

[23] Simon R. and Zurko M., "Separation of Duty in Role-Based Environments," *in Proceedings of the 10th Computer Security Foundations Workshop*, USA, pp. 562-568, 1997.

[24] Sindre G. and Opdahl A., "Eliciting Security Requirements with Misuse Cases," *Journal of Requirements Engineering*, vol. 10, no. 1, pp. 654-659, 2005.

[25] Sindre G. and Opdahl A., "Templates for Misuse Case Description," *in Proceedings of the 7th International Workshop on Requirements Engineering: Foundations for Software Quality*, Germany, pp. 77-79, 2002.

[26] Warmer J. and Kleppe A., *The Object Constraint Language: Precise Modeling with UML*, Addison Wesley, 1999.

**Khaled Alghathbar** PhD, CISSP, CISM, PMP, BS7799 Lead Auditor, is an associate professor and the director of the Centre of Excellence in Information Assurance in King Saud University, Saudi Arabia. He is a security advisor for several government agencies. His main research interest is in information security management, policies and design. He received his PhD in Information Technology from George Mason University, USA.