Specifying Usage Control Model With Object Constraint Language

Min Li[1]

[1]Australian Council for Educational Research

Abstract

The recent usage control model (UCON) is a foundation for next-generation access control models with distinguishing properties of decision continuity and attribute mutability. Constraints in UCON are one of the most important components that have involved in the principle motivations of usage analysis and design. The importance of constraints associated with authorizations, obligations, and conditions in UCON has been recognized but modeling these constraints has not been received much attention. In this paper we use a de facto constraints specification language in software engineering to analyze the constraints in UCON model. We show how to represent constraints with object constraint language (OCL) and give out a formalized specification of UCON model which is built from basic constraints, such as authorization predicates, obligation actions and condition requirements. Further, we show the flexibility and expressive capability of this specified UCON model with extensive examples.

Received on 09 January 2012; accepted on 11 January 2012; published on 04 February 2013

doi:10.4108/trans.sis.2013.01-03.e5

1. Introduction

Modern information systems require fine-grained and flexible access control policies, which need dynamic and expressive access control models. Traditional access control models such as lattice-based access control (LBAC) [19] and role-based access control (RBAC) [20] primarily consider static authorization decisions based on subjects' permissions on target objects. Policy-based authorization management systems have been proposed [6, 10, 13], in which a centralized reference monitor (or distributed reference monitor with centralized administration) checks a subject's permission when access is requested and the request is granted according to the security policies at the time of the access request. Once a subject is granted a permission, there are no more security checks for continued access. Developments in information technology, especially in electronic commerce applications, require additional features for access control. Recently proposed usage control (UCON) model is a new access control model that extends traditional access control models in multiple aspects [17] and considered as the next generation access control model [21].

The usage control (UCON) model was introduced as a unified approach to capture a number of extensions for traditional access control models. In the UCON model, the authorization-based decision process utilizes subject attributes and object attributes. Attributes can be identities, security labels, properties, capabilities, and so on. The UCON model includes obligation and conditions as well as authorizations as part of usage decision process to provide a richer and finer decision capability. Obligations are requirements that have to be fulfilled for usage allowance. Conditions are subject and object-independent environmental requirements that have to be satisfied for access. These decision predicates can be evaluated before or during exercise of a request. In addition, usage of target object may require certain updates on subject or object attributes before, during or after a usage exercise.

Park and Sandhu [17, 21] presented the conceptual model of UCON, which consists of several constraints. For example, people may have to click 'accept' button for license agreement or have to fill out a certain form to download a company's whitepaper. In addition, there are environmental requirements, such as, only IEEE member can access full papers in the IEEE digital library. Constraints can be described in natural languages, such as English, or in more formal languages. Natural language specification has the

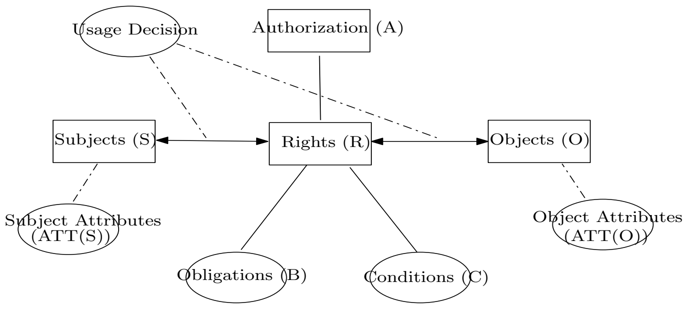

Figure 1. Components of UCON model

advantage of ease of comprehension by human beings, but may be prone to ambiguities [1]. Constraints in formal language are suitable for persons with a strong mathematical background, but difficult for average business or system developers to use. For instance, Zhang et al.[23] proposed a formalized specification of the principles of UCON with a temporal logic. The authors in [23] are security experts and system developers who have to understand organizational objectives and articulate major policy decisions.

This paper focuses on constraints specification, that is how constraints can be represented. The main contribution of this paper is to specify constraints of UCON model with object constraints language (OCL). Although constraints are one of the important components of UCON model, there is less study in previous works stressing this. With OCL, we provide a tool to precisely describe constraints for system designers and administrators. The specification also provides the precise meaning of the new features of UCON, such as the mutability of attributes and the continuity of usage control decisions. Another contribution of this paper is that we give out a formalized specification of UCON model which is built from these basic constraints, such as authorization predicates, obligation actions and condition requirements.

The rest of this paper is organized as follows. In the next section, we identify the motivation of our work in this paper and review the related technologies such as UCON, Unified Modeling Language (UML) and OCL. Constraints in authorization decisions, obligations and conditions are discussed in section 3. Formalized specifications of usage control model are expressed with OCL in section 4. Some related work is reviewed in section 5 and the conclusions are in section 6.

2. Motivation and Related Technologies

Constraints in UCON are one of the most important components that have involved in the principle motivations of usage analysis and design. Using OCL that has been used to express constraints in analysis and design as an industrial standard constraints specification language, we demonstrate that OCL can help us specify previously identified constraints at the system design step.

2.1. Usage Control

UCON has recently received considerable attention as a promising alternative to traditional access control model, such as access matrix [16], mandatory access controls (MAC) [2, 11], discretionary access control (DAC) and role-based access control (RBAC) [12, 22]. Usage control is used for the access control in the pervasive environment. There are eight components: subjects, subject attributes, objects, object attributes, rights, authorizations, obligations, and conditions in usage control model (see Fig. 1). Subjects and Objects are familiar concepts from traditional access control, and are used in their familiar sense in this paper. A right represents access of a subject to an object, such as read or write. Subject and object attributes are properties that can be used during the access decision process. Examples of subject attributes are identities, group names, roles, memberships, credits, etc. Examples of object attributes are security labels, ownerships, classes, access control lists, etc. In an online shop a price could be an object attribute, for instance, a particular e-book may stipulate a 10 price for a 'read' right and a 15 price for a 'print' right.

Authorizations, obligations and conditions are decision factors employed by the usage decision functions to determine whether a subject should be allowed to access an object with a particular right. In addition to these three decision factors, modern information system require two other important properties called 'continuity' and 'mutability' as shown in Fig. 2. In traditional access control, authorization is assumed to be done before access is allowed (pre). However, it is quite reasonable to extend this for continuous enforcement by evaluating usage requirements throughout usages (ongoing). the presence of ongoing decisions is called the continuity of UCON. Mutability means that one or more subject or object attribute values can be updated as side-effects of subjects' actions. In case, attributes are mutable, updates can be done either before (pre), during (ongoing) or after (post) usages shown in Fig. 2.

2.2. Unified Modeling Language and Object Constraints Language

The UML [18] is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of system analysis and design and further software implementation. The UML has become a standard modeling language in the filed of software engineering.

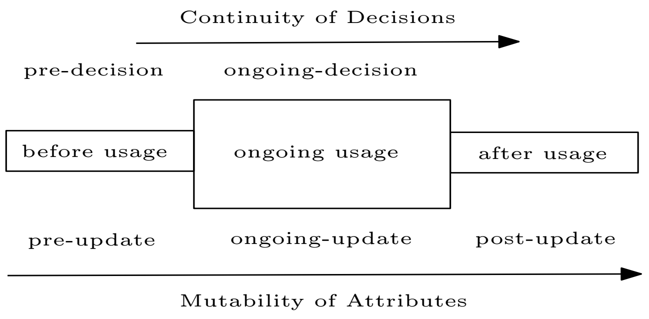

Figure 2. Continuity and mutability properties of UCON

Expression with OCL are described with the context of an instance of a specific type. In an OCL expression, the reserved word **self** is used to refer to the contextual instance. The type of the context instance of an OCL expression is written with the **context** keyword, followed by the name of the type. The label **invar**: declares the constraint to be an invariant constraint. For example, suppose that employees work for a company and they are involved in projects. These relationships can be modeled using the class model of the UML. If the context is Company, then self refers to an instance of Company. The following shows an example of OCL constraint expression describing a company that has more than 200 employees:

context Company **invar**:
self.employee → size > 200

The self.employee is a set of employees that is selected by navigating from Company class to Employee class though an association. The "." stands for a navigation. A property of a set is accessed by using an arrow "→" followed by the name of the property. A property of the set of employees is expressed using a keyword 'size' in this example.

An OCL expression delivers a subset of a collection. That is, the OCL has special constructs to specify a selection from a specific collection. For example, the following OCL expression specifies that the collection of employees whose age is over 50 is not empty:

context Company **invar**:
self.employee → select(age > 50) → notEmpty

The **select** takes an employee from self.employee and evaluates an expression (age > 50) for the employee. If this evaluation result is true, then the employee is in the result set. More examples can be reviewed in [18].

3. Constraints in UCON

Constraints are an important aspect of access control and are a powerful mechanism for laying out a higher-level organizational policy. Consequently the specification of constraints needs to be considered. This issue has received surprisingly little attention in the research literature. Next we will give out the main constraints in UCON.

Authorization Constraints

In today's highly dynamic, authorizations are predicates based on subject and/or object attributes, which determine whether a subject should be allowed to access an object with particular right. Before authorization, predicates on attributes have to be satisfied. A predicate is a boolean-valued polynomially computable function built from a set of a subject *s*'s and an object *o*'s attributes and constraints. The following examples show how we can specify this type of constraints using OCL.

Example 1*: The subject's credit attribute value in current state of the system should be larger than $100.*

context State **invar**:
self.attribute → subject.credit > $100

Obligation Constraints

In UCON, an obligation is an action that must be performed by a subject before or during an access, such as, filling out a form before playing a licensed music file.

Example 2*: The downloading of a music file may need the requesting subject to click a privacy button.*

context Downloading **invar**:
self.subject.click'privacy' = true

Condition Constraints

Conditions are environmental restrictions that have to be valid before or during a usage process, such as system clock, location, system code, etc.

Example 3*: A subject obtains a permission only when the system clock is in daytime.*

context Obtain a permission **invar**:
self.systemclock.daytime = true

Mutability Constraints

Mutability means that subject and/or object attributes can be updated as the results of an access. There are three kinds of updates: pre_updates, on_updates, and post_updates. The updating of attributes as side-effect of subject activity is a significant extension of classic access control where the reference monitor mainly enforces existing permissions.

Example 4*: The subject's current usage number of an object is increased by 1 at the time of access and decreased by 1 at the end of access. The update of the object's attribute 'usageNum' can be described as follow:*

context Attributes(o) **invar**:
self.pre_update → exists(usageNum = 'usageNum(o)+1')
self.post_update → exists(usageNum = 'usageNum(o)−1')

4. Specifying Usage Control Model with OCL

In this section we present a formalized specification of usage control models which is built from the basic components, such as authorization predicates and mutable attributes etc.

4.1. $UCON_{preA}$ – pre-authorization Models

Authorizations have been considered as the core of access control and extensively discussed since the beginning of access control discipline. Traditionally, access control research has focused on pre-authorizations in which a usage decision is made before a requested right is exercised. UCON$_{preA}$ models utilize these pre-authorizations for their usage decision processes. In UCON$_{preA}$ models, an authorization decision process is done before usage is allowed. We begin with the UCON$_{preA_0}$ which allows no updates of attributes.

```
context $preA_0$ invar:
init: self.access = ‘requesting’
derive: if self.access = ‘permitaccess’
            then   self.$preA(attr(s), attr(o), r)$ = true
            else   self.access = ‘denyaccess’
            endif
```

while *reqesting* means the access has been generated and is waiting for the system's usage decision, where *preA* is a functional predicate that utilizes $attr(s)$, $attr(o)$, and right r for usage decision making. We write ‘permitaccess’ to indicate that subject s is allowed right r to object o. Else, ‘denyaccess’ indicates that the system rejects the access request.

The UCON$_{preA_1}$ model is similar to UCON$_{preA_0}$ except it takes pre_update attributes into account, i.e., let ‘attr:string = self.pre_update → size()’ in the expression. We use the *size* property on the set of pre_updated attributes in $preA_1$, where ‘size()$\geqslant 1$’ indicates at least subject's or object's attributes are pre_updated.

```
context $preA_1$ invar:
init: self.access = ‘requesting’
derive: let attr:string = self.pre_update → size() in
          if self.access = ‘permitaccess’
          then
            self.$preA(attr(s), attr(o), r)$ = true and
            self.pre_update → size()$\geqslant 1$
          else    self.access = ‘denyaccess’
          endif
```

The specification of UCON$_{preA_3}$ is similar to that in UCON$_{preA_1}$ except it adds post_update processes, i.e., let ‘attr:string = self.post_update → size()’ in the expression.

***Example 5**: In a DRM pay-per-use application, a read access can be approved when the user Alice's credit is more than an ebook's value. Before the access can be begin, an update to Alice's credit is performed.*

```
context $preA_1$ invar:
init: self.access = ‘application’
derive: let attr:string = self.pre_update → size() in
if self.access = ‘read’
then
  self.attribute → $Alice.credit \geq ebook.value$ and
  self.pre_update → exists ($credit = Alice.credit - ebook.value$ → size()$\geqslant 1$)
else  self.access = ‘denyaccess’
endif
```

4.2. $UCON_{onA}$ – ongoing-authorization Models

In $UCON_{onA}$ model, ongoing-authorizations have been seldom discussed in access control literature. By utilizing ongoing-authorization, monitoring is actively involved in usage decisions while a requested right is exercised. This kind of continuous control is useful for relatively long-lived usage rights. In $UCON_{onA}$, there are four detailed models. $UCON_{onA_0}$ is immutable ongoing-authorization model that has no update procedure included. $UCON_{onA_1}$ is ongoing-authorization model with an optional pre-updates. UCON$_{onA_2}$ and UCON$_{onA_3}$ include ongoing updates and post updates respectively.

```
context $onA_0$ invar:
init: self.access = ‘accessing’
derive: if self.$onA(attr(s), attr(o), r)$ = false
            then    self.access = ‘revokeaccess’
            else    self.access = ‘endaccess’
            endif
```

$UCON_{onA}$ model introduces *onA* predicate instead of *preA*. Since there is no pre-authorization , the requested access is always allowed. The ‘accessing’ means that the system has permitted the access and the subject has been accessing the object immediately after that. In case certain attributes are changed and requirements are no long satisfied, ‘revoke’ procedure is performed. We write ‘revokeaccess’ to indicate that right r of subject s to object o is revoked and the ongoing access terminated. Else, ‘endaccess’ indicates that a subject finishes the usage and ends the access.

The expressions of onA_1, onA_2 and onA_3 are similar to that in onA_0 except they add the updating of attributes in the expression, i.e., pre_update, on_update, post_update, respectively. Here, for simplicity we just give out the description of onA_1 as following.

```
context $onA_1$ invar:
init: self.access = ‘accessing’
derive: let attr:string = self.pre_update → size() in
```

```
if self.$onA(attr(s), attr(o), r)$ = false
then
  self.access = 'revokeaccess' and
  self.pre_update → size()$\geqslant 1$
else    self.access = 'endaccess'
endif
```

Example 6*: Considering a limited number of simultaneous usages, each new access request must be granted and there is only one access generated from a single user at any time. when a new request is generated, one existing user's ongoing access is revoked by longest idle time. The policy can be specified as a combination policy of* onA_1, onA_2 *and* onA_3 *as follows.*

```
context $onA$ invar:
let attr:string = self.update → size() in
init: self.access = 'permitaccess'
derive:
  (1) self.pre_update → exists (accessingS = $accessingS(o) \cup \{s\}$)
  self.pre_update → exists (idleTime = 0)
  (2)   if self.access = 'accessing ∧ idle'   then
  self.on_update → exists (idleTime = $idleTime(s) + 1$)
  (3)   if   self.attributes   →   subject.startTime=
$Max_{idleTime}(object.accessingS)$
  then   self.access = 'revokeaccess'   and
  self.post_update → exists (accessingS = $accessingS(o) - \{s\}$)
  endif
```

where $Max_{idleTime}(object.accessingS)$ is the largest $idleTime$ in the object's $accessingS$ attribute. The first description is a onA_1 rule specifying that whenever a subject tries to access the object, there must be two pre-update before the subject starts to access. The second rule says that the mutability of the subject attribute by saying that there must be a continuous update of $idleTime$ whenever the status of subject is idle. The third rule specifies the revocation is is determined by the $idleTime$, and the attribute $accessingS$ is updated by removing the subject.

4.3. $UCON_{preB}$ – pre-obligations Models

$UCON_{preB}$ introduces pre-obligations that have to be fulfilled at the time of a request and before access is allowed. $UCON_{preB}$ models consist of two steps. First step is to select required obligation elements for the requested usage. The $getPreOBL$ function represents the pre-obligations required for s to gain r access to o. Second step is to evaluate whether the selected obligation elements have been fulfilled without any error (e.g., invalid e-mail addresses). The $preFulfilled$ predicate tells us if each of the required obligations in true. In $UCON_{preB}$ models, a request may require multiple pre-obligation elements to be fulfilled. Suppose the set of pre-obligation elements is indicated by M which is based on requests that consist of s, o and r.

```
context $preB_0$ invar:
let M: Set= $\{getPreOBL(s, o, r)\}$ in
init: self.access = 'requesting'
derive: if self.access = 'permitaccess'
then M → select($m|self.preFulfilled = false$) → is empty
else self.access = 'denyaccess'
endif
```

The expression (M → select($m|self.preFulfilled = false$) → is empty) indicates that all the required pre-obligation elements are fulfilled by using $preFulfilled$.

The specification of UCON$_{preB_1}$ is similar to that in UCON$_{preB_0}$ except that an pre_update action must be performed before 'permitaccess', i.e., let 'attr:string = self.pre_update → size()' in the expression.

```
context $preB_1$ invar:
let M: Set= $\{getPreOBL(s, o, r)\}$ in
init: self.access = 'requesting'
derive:
let attr:string = self.pre_update → size() in
if self.access = 'permitaccess'
then    M → select($m|self.preFulfilled = false$) → is
empty   and
self.pre_update → size()$\geqslant 1$
else    self.access = 'denyaccess'
endif
```

The UCON$_{preB_3}$ model is similar to UCON$_{preB_0}$ except it adds post_update processes.

Example 7*: In an online electronic marketing system, in order to place an order, a customer has to click a button to agree to the order policies. We define an action click_agreement as an obligation for each other, where the obligation subject is the same as the ordering subject, and the agree_statement is the obligation object. A customer's* $orderList$ *is updated by adding the ordered item after he/she places an order. This can be expressed with a* $preB_3$ *policy as the following.*

```
context $preB_3$ invar:
let M: Set= $\{(s, agree\_statement, order)\}$ in
init: self.access = 'requesting'
derive: if self.access = 'permitaccess'
then M → select($m|self.preFulfilled = false$) → is empty
self.post_update → exists ($orderList = orderList(s) \cup \{o\}$)
else self.access = 'denyaccess'
endif
```

4.4. $UCON_{onB}$ – ongoing-obligations Models

$UCON_{onB}$ models are similar to $UCON_{preB}$ models except that obligations have to be fulfilled while rights are exercised. Ongoing-obligations may have to be fulfilled periodically or continuously. In $UCON_{onB}$ models, there are four detailed models based on mutability issues. $UCON_{onB_0}$ includes ongoing-obligation predicate instead of pre-obligations predicate. $UCON_{onB_1}$, $UCON_{onB_2}$, and $UCON_{onB_3}$ are same as $UCON_{onB_0}$ except that they add pre-updates, ongoing-updates, and post-updates, respectively.

```
context $onB_0$ invar:
let M: Set= $\{getOnOBL(s, o, r)\}$ in
init: self.access = 'accessing'
derive:
if M → select($m|self.onFulfilled = false$) → notempty
then self.access = 'revokeaccess'
else self.access = 'endaccess'
endif
```

Similar to *preB*, the set *M* shows the selection of required ongoing-obligation elements. The specification (M → select($m|self.onFulfilled = false$) → notempty) indicates that not all required ongoing-obligation elements are fulfilled by using *onFulfilled*.

The expressions of onB_1, onB_2 and onB_3 are similar to that in onB_0 except it adds the updating of attributes in the expression, i.e., pre_update, on_update, post_update, respectively. Next, the description of onB_1 is given for simplicity.

```
context $onB_1$ invar:
let M: Set= $\{getOnOBL(s, o, r)\}$ in
init: self.access = 'accessing'
derive:
let attr:string = self.pre_update → size() in
if M → select($m|self.onFulfilled = false$) → notempty
then    self.access = 'revokeaccess' and
self.pre_update → size()$\geqslant 1$
else    self.access = 'endaccess'
endif
```

4.5. $UCON_{preC}$ – pre-conditions Model

Conditions are environmental restrictions that have to be satisfied for usages. By utilizing conditions in usage decision process, $UCON_{preC}$ can provide fine-grained controls on usage. unlike authorization and obligation models, condition models cannot be mutable. $UCON_{preC}$ introduces pre-conditions predicate that has to be evaluated before requested rights are exercised.

```
context $preC_0$ invar:
let M: Set= $\{getPreCON(s, o, r)\}$ in
init: self.access = 'requesting'
derive: if self.access = 'permitaccess', then
M → select($m|self.preConChecked = false$) → is empty
else self.access = 'denyaccess'
endif
```

In this specification, a set of relevant condition elements *M* is selected based on a request possibly using subject or object attributes. To allow a request, all of the selected condition restrictions have to be evaluated by using *preConChecked*.

4.6. $UCON_{onC}$ – ongoing-conditions Model

In many cases, environmental restrictions have to be satisfied while rights are in active use. This could be supported within $UCON_{onC}$ model. In $UCON_{onC}$, usages are allowed without any decision process at the time of requests. However, there is an ongoing-conditions predicate to check certain environmental status repeatedly throughout the usages.

```
context $onC_0$ invar:
let M: Set= $\{getOnCON(s, o, r)\}$ in
init: self.access = 'accessing'
derive:
if M → select($m|self.onConChecked = false$) → notempty
then self.access = 'revokeaccess'
else self.access = 'endaccess'
endif
```

5. Related Work

The development of access control models has experienced a long history. There are two main approaches in this field. One is about traditional access control models, which have been discussed in the introduction. The other approach is about the research of temporal access control models, which introduce the temporal attributes into traditional access control with temporal logic. A temporal authorization model for database management systems was first proposed by Bertino et al. [3–5]. In this model, a subject has permissions on an object during some time intervals or a subject's permission is temporally dependent on an authorization rule. For example, a subject can access a file only for one week. Our specified model is different: we consider the temporal characteristics in a single-usage period, with mutable attributes of subject and object before, during, and after an access, that is, the temporal properties are result of mutability of subject and object attributes, which change due to the side effects of access and usages.

Joshi et al. [15] presented a generalized temporal RBAC model (GTRBAC) to specify temporal constraints in role activation, user-role assignment, and role-permission assignment. For example, a user can only activate a role for a particular duration. The concept of temporal constraint is different from the mutability constraints of UCON, since it does not have update actions. The dependency constraint in GTRBAC [14] is similar to the concept of obligation in UCON, but the dependency is more like the implication relation between events in GTRBAC, i.e., if an event happens, it triggers another event; while in UCON, obligations are explicit required actions to permit an access.

Bettini et al. [7, 8] presented concepts of provisions and obligation in policy management: provisions are conditions or actions performed by a subject before authorization decision, while obligations are conditions or actions performed after an access. In this paper, we distinguish between conditions and obligations. All the actions that a subject has to perform before usage are regarded as obligations, while for future actions, we consider them as the obligations for future usage requests or long-term obligations. Chomicki and Lobo [9] investigate the conflicts and constraints of historical actions in policies. In their paper, actions are applications activities and constraints are expressed with linear-time temporal connectors. In our paper, we specify obligations as actions required by an access and give formal specification with OCL.

6. Conclusions

This paper has discussed the constraints in UCON model and provide various kinds of constraints representation with object constraint language. We have analyzed the constraints in UCON such as decision actor constraints and mutability constraints etc. We also provide a tool to precisely describe constraints for system designers and administrators. Furthermore, we give out a formalized specification of UCON model which is built from these basic constraints, such as authorization predicates, obligation actions and condition requirements etc. We show the flexibility and expressive capability of this model by specifying the core models of UCON with extensive examples.

References

[1] G. Ahn and M. Shin: Role-Based Authorization Constraints Specification Using Object Constraint Language. Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (2001) 157-165

[2] D. E. Bell and L. J. Lapadula, Secure Computer Systems: Mathematical Foundations and Model. Mitre Corp. Report No.M74-244, Bedford, Mass., 1975.

[3] E. Bertino, C. Bettini, and P. Samerati, A temporal authorization model. In Proceedings of ACM Conference on Computer and Communication Security. ACM, New York, 1994.

[4] E. Bertino, C. Bettini, and P. Samerati, A temporal access control mechanism for database systems. IEEE Transactions on Knowledge and Data Engineering 8, 1 (Feb.) 1996.

[5] E. Bertino, C. Bettini, and P. Samerati,An access control model supporting periodicity constraints and temporal reasoning. ACM Transaction on Database Systems 23, 3 (Sept.)1999.

[6] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca, A Logical Framework for Reasoning about Access Control Models, In Proc. of Sixth ACM Symposium on Access Control Models and Technologies, 2001.

[7] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera, Obligation monitoring in policy management. In Proceedings of the 3rd InternationlWorkshop on Policies for Distributed Systems and Networks, 2002.

[8] C. Bettini, S. Jajodia, X. S. Wang, and D. Wijesekera, Provisions and obligations in policy management and security applications. In Proceedings of the 28th VLDB Conference, 2002.

[9] J. Chomicki, and J. Lobo, Monitors for history-based policies. In Proceedings of the 2nd Internationl Workshop on Policies for Distributed Systems and Networks, 2001.

[10] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, The Ponder Policy Specification Language, In Proc. of the Workshop on Policies for Distributed System s and Networks, 2001.

[11] D. E. Denning, A lattice model of secure information flow, Communications of the ACM, vol. 19, no. 5, 1976.

[12] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. Richard Kuhn and R. Chandramouli, Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security, Volume 4, Number 3, August 2001.

[13] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian, Flexible Support for Multiple Access Control Policies, ACM Transactions on Database Systems, June, 2001

[14] J. Joshi, E. Bertino, B. Shafiq, and A. Ghafoor, Constraints: Dependencies and separation of duty constraints in gtrbac. In Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, 2003.

[15] J. Joshi, E. Bertino, B. Shafiq, and A. Ghafoor, A generalized temporal role-based access control model. IEEE Transactions on Knowledge and Data Engineering 17, 1, 2005.

[16] M. H. Harrison, W. L. Ruzzo, and J. D. Ullman, Protection in Operating Systems, Communication of ACM, Vol 19, No. 8, 1976.

[17] J. Park and R. Sandhu, The UCONABC Usage Control Model, ACM Transactions on Information and Systems Security, Feb., 2004.

[18] M. Richters and M. Gogolla: On Formalizing the UML Object Constraint Language OCL. In Tok-Wang Ling etc editor: 17th International Conference on Conceptual Modeling (ER). Vol. 1507 Springer-Verlag (1998) 449-464

[19] R. Sandhu, Lattice-Based Access Control Models, IEEE Computer, Vol.26, No.11, November 1993.

[20] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, Role Based Access Control Models, IEEE Computer, 29, (2), pp.38-47, 1996

[21] R. Sandhu and J. Park, Usage Control: A Vision for Next Generation Access Control, The Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security, 2003.

[22] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, Role-Based Access Control Models, IEEE Computer, Volume

29, Number 2, February 1996.

[23] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu, A Logical Specification for Usage Control, In Proc. of the 9th ACM Symposium on Access Control Models and Technologies, 2004.