A Logical Specification for Usage Control*

Xinwen Zhang
George Mason University
xzhang6@gmu.edu

Jaehong Park
George Mason University
jpark2@gmu.edu

Francesco Parisi-Presicce
George Mason University
fparisip@gmu.edu

Ravi Sandhu
George Mason University and NSD Security
sandhu@gmu.edu

ABSTRACT

Recently presented usage control (UCON) has been considered as the next generation access control model with distinguishing properties of decision continuity and attribute mutability. A usage control decision is determined by combining authorizations, obligations, and conditions, presented as $UCON_{ABC}$ core models by Park and Sandhu. Based on these core aspects, we develop a first-order logic specification of UCON with Lamport's temporal logic of actions (TLA). The building blocks of this model include: (1) a sequence of states expressed by attributes of subjects, objects, and the system, (2) state predicates on subject and object attributes, (3) pre-defined authorization actions performed by the security system and subjects, (4) obligation actions, and (5) condition predicates on system attributes. For a UCON model we define a set of temporal logic formulas that hold as usage control policies. We show the flexibility and expressive capability of this logic model by specifying the new features and core models of UCON.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access controls*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized access*

General Terms

Security

Keywords

access control, usage control, security policy, logic specification

1. INTRODUCTION

Traditional access control models such as lattice-based access control (LBAC) [1, 20] and role-based access control (RBAC) [21] primarily consider static authorization decisions based on subjects' permissions on target objects. On the other side, policy-based authorization management systems have been recently proposed and developed by researchers [5, 10, 13, 14]. In such systems there is a centralized reference monitor (or distributed reference monitor with centralized administration) that checks each user's permission when he/she requests access. A permission is granted to a subject according to the security policies at the time of the access request. Once a subject is granted a permission, he/she can access the object as many times as he/she likes.

The development in information technology, especially in electronic commerce applications, requires additional features for access control. In recent information systems, usage of a digital object can be not only time-independent like read and write, but also temporal and time-consuming, such as payment-based online reading, metered by reading time or chapters, or a downloadable music file that can only be played 10 times. So a subject's permission may decrease, expire, or be revoked along with the usage of the object.

Recently proposed usage control (UCON) is a new access control model that extends traditional access control models in multiple aspects. In UCON, an access may be an instantaneous action, but may also be a process lasting for some duration with several related and sequent actions. An access decision can be made before or during the access process, or both. Actions during an access period possibly result in changes to subject or object attributes. A usage decision in UCON is made by policies of authorizations, obligations, and conditions (also referred as $UCON_{ABC}$ core models). Authorization decisions are determined by policies using attributes of the subject, object, and right. Obligations are actions that are required to be performed before or during the access process. Conditions are environment restrictions that are required to be valid before access or during access. An extreme example of UCON is the traditional access control models, in which the authorization decision is made instantly when an access request is generated, and there is no further check after that. More generally, usage control is a comprehensive model to represent the underlying mechanism of existing access control models and policies, as well as the access control in DRM, trust management, and other modern information systems.

The distinguishing properties of UCON beyond traditional access control models are continuity of access decision and mutability of subject attributes and object attributes. In UCON, authorization decisions are not only checked and made before the access, but may be repeatedly checked during the access and may revoke the access if some policies are not satisfied, according to the changes of the subject or object attributes, or environmental conditions. Mutability is a new concept introduced by UCON, but its features can be found in traditional access control models and policies. For ex-

*This work is partially supported by the National Science Foundation.

SACMAT'04, June 2–4, 2004, Yorktown Heights, New York, USA.

ample, in a Chinese Wall policy, if a subject accesses an object in a conflict-of-interest set, then he/she cannot access any other conflicting objects in the future. That means, the potential object list that the subject can access (we can consider this a subject attribute) has been changed as a side-effect of his/her previous access. This change, consequently, will restrict the next access of this subject. History-based access control policies can be expressed by UCON with this feature of attribute mutability. Also, mutability is useful to specify dynamic constraints in access systems, such as separation of duty (SOD) policies, cardinality constraints, etc. Another prospective area is consumable access. Consumable access is becoming an important aspect in many applications, especially in digital right management (DRM). For example in a pay-per-use DRM application with fixed credit of a subject, the available access time decreases with ongoing access.

Continuity and mutability in UCON introduce interactive and concurrent concepts into access control. An access results in the update of subject or object attributes as side-effects. These changes, on the other hand, will result in the change of other ongoing or future accesses by the same subject, or to the same object, or some access that is implicitly related. That means, an access may change not only its own state, but also the state of other accesses.

Park and Sandhu [18, 19] presented the conceptual model of UCON, which consists of several core sub-models including authorization, obligations, and conditions. To illustrate the fundamental principle of UCON and its powerful and flexible expressive capability, we present in this paper a logic specification with Lamport's temporal logic of actions (TLA) [15]. The basic components of this logic model are predicates between subject, object, and system attributes, as well as some predefined actions during the access period by the system or by subjects. A usage control policy is a logic formula built from these components. We regard obligations as actions separate from the authorization actions. The conditions of UCON are specified by predicates on the system attributes.

The rest of this paper is organized as follows. Section 2 shows a motivating example of usage control, especially the new features of continuity and mutuality. Section 3 gives a brief introduction of UCON, and the core authorization sub-models. Section 4 introduces TLA briefly. Section 5 presents the details of our logic model. Section 6 presents the specification of the core authorization models with our logic model. Section 7 introduces the specification of obligations and conditions. Some related work in access control with temporal aspects is reviewed in Section 8. Finally, we summarize this paper and present some ongoing and future work in Section 9.

2. MOTIVATING EXAMPLE

In this section we present an example motivating the new features of UCON. Traditional access control models and policies have difficulties, or lack the flexibility to specify policies in these scenarios. This example is originally from pervious papers [18, 19], but we describe the problem and policies in more detail.

Example 1 Suppose in a DRM application with limited number of simultaneous usages, an object o can only be accessed and used by 10 users at a time. Each new access request is allowed. We assume that there is only one access generated from a single user. If the number of users accessing the object is 10, then one existing user's ongoing access is revoked when a new request is generated. There are different policies to determine which user's ongoing access must be stopped as follows.

(a) Revocation by start time: the longest usage will be revoked.

(b) Revocation by idle time: the usage with the longest idle time will be revoked.

(c) Revocation by total usage time: the user with the longest accumulating usage time will be revoked.

For these three different policies, we need to define different temporal attributes for subjects and objects. Specifically:

(a) There are two object attributes: the number of total ongoing accesses and a list of accessing subjects. For each subject, we define the starting time as an attribute. Each time a new access request is generated, the set of accessing subjects is updated by adding the requesting subject. In UCON terminology, this is a pre-update. If the total accessing number is already 10, then the ongoing subject with the earliest start time is revoked, and the new access is permitted. When an access is ended by a subject or revoked by the system, the total accessing number is updated by subtracting one, and the subject is removed from the accessing list. This is a post-update.

(b) Objects have the same attributes as in (a). Each subject has two attributes: the state of the subject with a value of $busy$ or $idle$, and continuous idle time in a single usage process. In order to monitor the idle time, the system has to check the status and update the idle time during the entire ongoing access by means of ongoing-update. Similar to (a), there are pre-update, revoking access, and post-update actions, in which the revoking action is performed to the longest idle access.

(c) Here again objects have the same attributes as in (a). Each subject has an attribute of accumulating usage time to record the total usage time of this subject on this object over the subject and object life. Similar to (a) and (b), there are pre-update, revoking access, and post-update. In addition, there is a post-update of subject attribute after the usage (either ended by a subject or revoked by the system) to add the time of this usage with historically accumulating usage time.

In this example, an access is like a process that interacts with other related processes which are accessing or trying to access the same object concurrently. An access decision is no longer a single function of (subject, object, right), but may depend on access(es) from other subjects, and may change the status of other accesses.

There are many other examples to motivate UCON model that cannot be expressed by traditional access control models. We will explore some of these later in this paper as we describe the logical language.

3. USAGE CONTROL

In this section we briefly review the general ideas of UCON and the core authorization models. The details of these models can be found in [18, 19].

As depicted in Fig. 1, a usage control system has six components: subject and attributes, object and attributes, rights, authorizations, obligations, and conditions[1]. The authorization, obligations and conditions are components of usage control decisions. An authorization rule permits or denies access of a subject to an object with a specific right based on subject and object attributes. Obligations are activities that are performed by subjects or by the system. Conditions are system environment restrictions, not related to subject or object attributes. The most important properties that distinguish

[1]Note that this diagram is slightly different from that in [18, 19]. Here we place the usage decisions at the center, whereas in [18, 19] the rights are in the center.

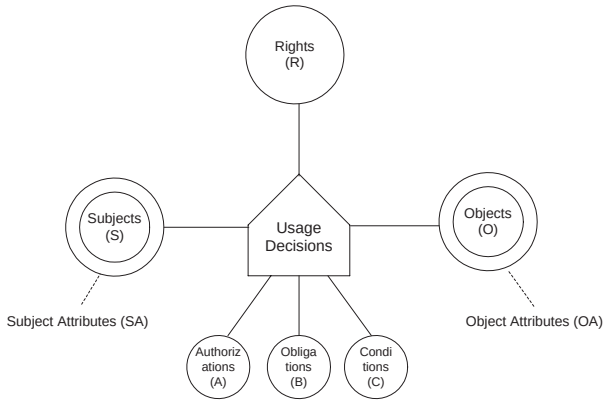

Figure 1: Usage control model

UCON from traditional access control models and trust management are the continuity of usage decisions and the mutability of attributes. Fig. 2 shows a complete usage process consisting of three phases along time series: before usage, ongoing usage, and after usage. To enforce control decisions, we distinguish two different types: pre-decision and ongoing-decision. In the after-usage phase, we don't enforce any decision since there is no access control after a subject finishes his/her usage on an object[2].

For mutability, there are three kinds of updates along the three phases: pre-update, ongoing-update, and post-update. All these updates are performed and monitored by the security system. An update of subject or object attributes may result in a system action, not only on this usage, but also on other accesses related to the same subject or object. The actions on the current usage may generate cascading updates or other actions, while actions on other usages act as external events that would cause a change of those usages, such as access revocation. Based on the relationship between

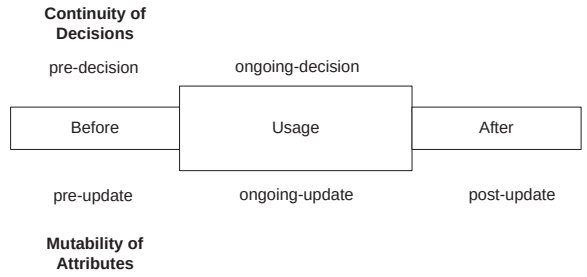

Figure 2: Continuity and mutability properties of UCON

authorization decisions and attribute mutability, Park and Sandhu [18, 19] defined seven core models summarized below.

- $preA_0$: A usage control decision is determined by authorization rules before access, and there is no attribute update before, during, or after usage.
- $preA_1$: A usage control decision is determined by authorization rules before access, and one or more subject or object attributes are updated before usage.
- $preA_3$: A usage control decision is determined by authorization rule before access, and one or more subject or object attributes are updated after usage.
- onA_0: Usage control is checked and the decision is determined by authorization rules during access, and there is no attribute update before, during, or after usage.
- onA_1: Usage control is checked and the decision is determined by authorization rules during access, and one or more subject or object attributes are updated before usage.
- onA_2: Usage control is checked and the decision is determined by authorization rules during access, and one or more subject or object attributes are updated during usage.
- onA_3: Usage control is checked and the decision is determined by authorization rules during access, and one or more subject or object attributes are updated after usage.

Note that in the case of authorization before access and update during usage, since the update of attributes will not trigger any authorization check during usage, it has the same effect as update after usage ($preA_3$)[3]. So this case is not included in UCON. For models which enforce authorization rules during usage, ongoing-checking captures not only the attribute changes from other related usage processes, but also the attribute changes because of this local usage process (onA_2).

These core authorization models as well as the obligations and conditions provide flexible and expressive capability for UCON. This makes UCON a very comprehensive access control model not only for traditional access control systems, but for more recently developed information systems. We will discuss obligations and conditions in Section 7.

4. TEMPORAL LOGIC OF ACTIONS

Extended from temporal logic [17] by introducing boolean valued actions, TLA [15] is a powerful tool to specify systems and their properties, especially for interactive and concurrent systems. In this section we first give a brief introduction of basic terms and the syntax of temporal formula. Then we will introduce some additional temporal operators and their semantics to be used in our logic language.

4.1 Building Blocks

Variables, values, and states are basic concepts in TLA. Values are data items, such as numbers 1, -5, string "abc", sets like the set N of natural numbers, etc. Note that booleans $true$ and $false$ are not considered as values in TLA. A variable has a name like x and y, and can be assigned a value. Normally we assume there is an infinite set of variables. A constant is a variable that is assigned a fixed value. A state is characterized by assignments of a value $s(x)$ to each variable x. The semantic meaning of $s(x)$ is denoted as $s[\![x]\!]$. Generally in TLA, a semantics is given by assigning a semantic meaning $[\![F]\!]$ to each syntactic object F.

A function is a nonboolean expression built from variables and constants, such as $x^2 + y - 3$. The semantic meaning $[\![f]\!]$ of a function f is a mapping from states to values. For example, $[\![x^2 + y - 3]\!]$ is the mapping that assigns to the state s the value $s[\![x]\!]^2 +$

[2]There can be obligations and conditions (post-obligation and post-conditions) defined in this phase. UCON is a session-based access control model, since it controls the current access request and ongoing access. The obligations and conditions after an access are regarded as long-term obligations and conditions, which are not included in UCON, but in an administrative model. In this paper we only focus on the core aspects of UCON, while an administrative model will be developed in the future.

[3]This assumes no "interference" with other ongoing accesses.

$s[\![y]\!] - 3$, where $s[\![x]\!]$ and $s[\![y]\!]$ denote the values that s assigns to x and y. Generally:

$$s[\![f]\!] \equiv f(\forall `v': s[\![v]\!]/v)$$

where $f(\forall `v': s[\![v]\!]/v)$ is the value by substituting $s[\![v]\!]$ for v. Semantically, a variable is also a function that assigns the value $s[\![x]\!]$ to the state s.

A predicate is a boolean expression built from variables and constants, such as $x = y+1$. The semantic meaning $[\![P]\!]$ of a predicate P is a mapping from states to booleans. A state s satisfies a predicate P iff $s[\![P]\!]$, the value of $[\![P]\!]$ in s, equals $true$.

An action is a boolean-valued expression formed from variables, primed variables, and constants, such as $x' = y+1$ and $x'-1 \notin y'$. Semantically, an action represents a relation between old states and new states, where unprimed variables refer to the old state and the primed variables refer to the new state. Formally, an action A is a function assigning a boolean $s[\![A]\!]t$ to a pair of states (s, t). For example, $x' = y+1$ equals to the boolean value of $t[\![x]\!] = s[\![y]\!]+1$. We say that (s, t) is an A step if $s[\![A]\!]t$ equals $true$. Generally:

$$s[\![A]\!]t \equiv A(\forall `v': s[\![v]\!]/v, t[\![v]\!]/v')$$

Since a predicate P is a boolean expression built from variables and constants, in TLA it is regarded as an special action without primed variables. A pair (s, t) is a P step iff $s[\![P]\!]$ is $true$.

For any action A, $Enabled\ A$ is a predicate that is $true$ for a state s iff it is possible to take an A step starting in s. Semantically:

$$s[\![Enabled\ A]\!] \equiv \exists t \in St : s[\![A]\!]t$$

where St is the set of states.
For example, for action A: $y = x'^2 + 1$,

$$Enabled\ A \equiv \exists c : y = c^2 + 1$$

4.2 Temporal Formulas and Semantics

The basic temporal operator is $\Box$ (always). The semantics of a temporal action is defined using the concept of *behavior*. A behavior σ in TLA is an infinite sequence of states $< s_0, s_1, s_2, ... >$ (a finite set of states can be regarded as infinite with identical states). With this idea, the semantics of an atomic formula with actions is defined as:

$$< s_0, s_1, s_2, ... > [\![A]\!] \equiv s_0[\![A]\!]s_1$$
$$< s_0, s_1, s_2, ... > [\![\Box A]\!] \equiv \forall n \geq 0 : s_n[\![A]\!]s_{n+1}$$

The same semantics can be defined for predicates since a predicate is a special form of action.

In TLA, a formula is built from predicates and actions with logical connectors and temporal operators. Recursively, a temporal formula is defined by the following grammar in BNF:

$$< formula >:\equiv < predicate > \mid < action >$$
$$\mid \neg < formula > \mid < formula > \wedge < formula >$$
$$\mid < formula > \vee < formula > \mid < formula > \rightarrow <$$
$$formula > \mid \Box < formula > \mid$$

A formula is an assertion about a behavior. The semantic value $\sigma[\![F]\!]$ of a formula F is a boolean value on a behavior σ. Formally:

$$< s_0, s_1, s_2, ... > [\![F]\!] \equiv s_0[\![F]\!]s_1$$
$$< s_0, s_1, s_2, ... > [\![\Box F]\!] \equiv$$
$$\forall n \geq 0 :< s_n, s_{n+1}, s_{n+2}, ... > [\![F]\!]$$

4.3 Extension of TLA

Originally there is only the "always" ($\Box$) operator introduced in TLA. Some other future operators are defined using this, such as "eventually" ($\Diamond$) and "infinitely often" ($\Box\Diamond$). The relationship between the "always" and the "eventually" operators can be expressed as:

$$\Diamond F \equiv \neg\Box\neg F$$

Based on the semantics of temporal actions and formulas, we can define other temporal operators and semantics similarly.

4.3.1 *The* "Next" *and* "Until" *Temporal Operator*

For a behavior $< s_0, s_1, s_2, ... >$, the semantics of the *Next* operator ($\bigcirc$)is defined as:

$$< s_0, s_1, s_2, ... > [\![\bigcirc F]\!] \equiv s_1[\![F]\!]s_2$$

Until ($\mathcal{U}$) is a binary operator. A formula $F\mathcal{U}G$ is $true$ if F is always $true$ until G is $true$ along the sequence of states. Semantically:

$$< s_0, s_1, s_2, ... > [\![F\mathcal{U}G]\!] \equiv$$
$$\exists i \geq 0 : \big(s_i[\![G]\!]s_{i+1} \wedge (0 \leq j \leq i \rightarrow s_j[\![F]\!]s_{j+1})\big)$$

Note that the semantics of $F\mathcal{U}G$ has no requirement on G for s_j and F for s_i and its following states, which is different from the "until" in the English language.

There is an equivalence between these temporal operators:

$$\Diamond F \equiv (F \vee \neg F)\mathcal{U}F$$

4.3.2 *Past Temporal Operators*

TLA only defines future temporal operators like $\Box$ and $\Diamond$. From traditional temporal logic there are past temporal operators to specify the properties during the past time compared to the current time point. For a behavior $< s_0, s_1, s_2, ... >$ in TLA, if we consider s_0 as the state at the current time point, then $s_1, s_2, ...$ are states of the future on the time series. We use the state sequence $..., s_{-2}, s_{-1}$ for states during the past time along this time series. Therefore, a behavior is a state sequence:

$$< ..., s_{-2}, s_{-1}, s_0, s_1, s_2, ... >$$

Based on this, we can define past temporal operators similar to the future ones: $\blacksquare$ (*Has-always-been*), $\blacklozenge$ (*Once*), $\ominus$ (*Previous*), $\mathcal{S}$ (*Since*). Semantically:

$$< ..., s_{-2}, s_{-1}, s_0, s_1, s_2, ... > [\![\blacksquare F]\!] \equiv$$
$$\forall n < 0 : s_n[\![F]\!]s_{n+1}$$
$$< ..., s_{-2}, s_{-1}, s_0, s_1, s_2, ... > [\![\blacklozenge F]\!] \equiv$$
$$\exists n < 0 : s_n[\![F]\!]s_{n+1}$$
$$< ..., s_{-2}, s_{-1}, s_0, s_1, s_2, ... > [\![\ominus F]\!] \equiv s_{-1}[\![F]\!]s_0$$
$$< ..., s_{-2}, s_{-1}, s_0, s_1, s_2, ... > [\![F\mathcal{S}G]\!] \equiv$$
$$\exists i < 0 : \big(s_i[\![G]\!]s_{i+1} \wedge (i < j < 0 \rightarrow s_j[\![F]\!]s_{j+1})\big)$$

Similar to the future operators, there are some equivalences between these past operators. For example,

$$\blacklozenge F \equiv \neg\blacksquare\neg F$$
$$\blacklozenge F \equiv F\mathcal{S}(F \vee \neg F)$$

5. LOGICAL MODEL FOR AUTHORIZATION IN UCON

In this section we propose a logical approach for UCON. First we present some building blocks of the logic model, then we specify the core models in UCON with our logic model.

5.1 Attributes and States

As in TLA, a state is an assignment of values to variables. In UCON, there are three different kinds of variables: subject attributes, object attributes, and system attributes.

UCON is an attribute-based access control model. The authorization policies are determined by subjects attributes, objects attributes, and rights[4]. A subject or object attribute is a variable of a specific datatype, which includes a set of possible values and operators to manipulate them. A state of a subject or an object is an assignment of values to attribute(s). The datatype of an attribute depends on what kind of attribute it is, such as group membership, role, security clearance, credit amount, etc.

System attributes are variables that are not related to a subject or an object directly, such as system clock, location, etc. We define a special system state to specify the status of a single access process (s, o, r). Specifically, the function $state(s, o, r)$ is a mapping from $\{(s, o, r)\}$ to $\{initial, requesting, denied, accessing, revoked, end\}$. The semantics of the $initial$ state is that the access (s, o, r) has not been generated, while $requesting$ means the access has been generated and is waiting for the system's decision; $denied$ means that the system has denied the access request according to the authorization policies before usage; $accessing$ means that the system has permitted the access and the subject has been accessing the object immediately after that. An access will go to $revoked$ state when its ongoing-access is revoked by the system, or it will go to an end state when a subject finishes the usage. We believe that this set of values is minimal and complete to specify the states during a usage process.

The attributes of subjects, objects, and the system, as well as the constants comprise the basic terms of our logical model in UCON.

5.2 Predicates

A predicate is a boolean expression built from variables and constants, including subject attributes, object attributes, and system attributes. According to different forms of the attributes, we define four types of predicates in UCON:

- Unary predicate of a subject or object attribute: $p(sa)$ or $p(oa)$.
  e.g.: $Alice.credit \geq \$100.00$
  $file1.clearance = `supersecure'$

- Binary predicate between a subject attribute and an object attribute: $p(sa, oa)$.
  e.g.: $dominate(Alice.cleareance, file1.classification)$, $dominate(Alice.credit, ebook.value)$.

- Binary predicate $in(a_1, a_2)$ between attributes, where the a_2 is a set of possible values of a_1.
  e.g.: $in(Alice.ID, file1.acl)$, where ID is a subject identification, $file1.acl$ is a set of subject IDs as $file1$'s access control list.

- Ternary predicate $permit(s, o, r)$, which is $true$ if a subject s can access an object o with r. This predicate is the result of an authorization decision by the system.

5.3 Actions

As we have mentioned, there are three types of variables in UCON: subject attributes, object attributes, and system attributes. We define different actions to change the states of these variables.

[4]Right attributes were not explored in the original UCON model, and we also do not consider it in this paper

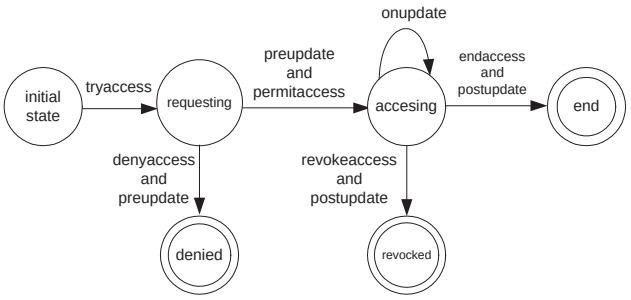

Figure 3: State transition with actions in a usage process

There are three actions defined in UCON that change the state of a subject or object: $preupdate$, $onupdate$, and $postupdate$. These actions are performed by the security system before, during, or after the access process. If the system performs the action successfully, the attribute value must be changed to a new value, and the action is $true$[5], otherwise, it is $false$. Note that in our specification we do not consider the time delay of an action. We assume that an action is always performed instantly. In a real implementation there is mechanism to monitor the process and audit the update, so that the system can recover if any failure happens.

For system states, we define a series of actions that change the status of an access (s, o, r). As mentioned before, there are six different possible values of $state(s, o, r)$ during a process life cycle. The transition from a state to another state is an action, as shown in Fig. 3.

Fig. 4 shows the actions that are performed by the system and by a subject during a usage. These actions are briefly explained below.

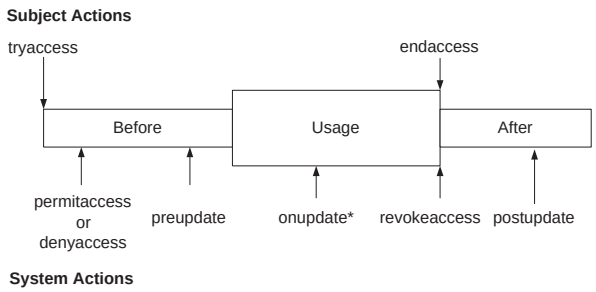

Figure 4: Subject and system actions

1. $tryaccess(s, o, r)$: generates a new access request (s, o, r), performed by a subject.

2. $permitaccess(s, o, r)$: grants an access request of (s, o, r), performed by the system.

3. $denyaccess(s, o, r)$: rejects an access request of (s, o, r), performed by the system.

4. $revokeaccess(s, o, r)$: revokes an ongoing access (s, o, r), performed by the system.

[5]Semantically, the action is a transition step from the original state with old value to new state with updated value. Here we simply say the action is $true$.

5. $endaccess(s, o, r)$: ends an access (s, o, r), performed by a subject.

6. $preupdate(attribute)$: updates a subject or object attribute before access, performed by the system.

7. $onupdate(attribute)$: updates a subject or object attribute during usage phase, performed by the system. The star symbol indicates that this action may be performed repeatedly by the system to continuously update an attribute.

8. $postupdate(attribute)$: updates a subject or object attribute after access, performed by the system.

Note that Fig. 3 only shows state transitions of the system attribute $state(s, o, r)$ in one usage session. It does not show subject/object attributes or other system attributes. Formally, a state in a usage process includes the states of subject attributes, object attributes, and system attributes.

5.4 Logical Formulas

A logic formula is built from state predicates and actions with logic connectors and temporal operators.

DEFINITION 1. *A logical formula in UCON is defined by the following grammar in BNF:*

$$\phi ::= a|p(t_1, ..., t_n)|(\neg\phi)|(\phi\wedge\phi)|(\phi \rightarrow \phi)|\forall x : \phi|\exists x : \phi|\Box\phi|\Diamond\phi|\bigcirc \phi|\phi\mathcal{U}\phi|\blacksquare\phi|\blacklozenge\phi|\ominus \phi|\phi\mathcal{S}\phi|$$

where a is an action, p is a predicate of arity n, t_1, ..., t_n are terms, and x is a variable.

5.5 Logical Model of UCON

DEFINITION 2. *An logic authorization model of UCON is a triple:$\mathcal{M} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, where*

- *$\mathcal{S}$ is a sequence of states of subject, object, and the system attributes,*
- *$\mathcal{P}$ is a finite set of state predicates on subject and/or object attributes,*
- *$\mathcal{A}$ is a finite set of state actions.*

If a model $\mathcal{M}$ with a state s satisfies a formula ϕ, we write $\mathcal{M}, s \vDash \phi$. Semantically,

1. $\mathcal{M}, s_0 \vDash p$ iff $s_0[\![p]\!]$, where $p \in P$.
2. $\mathcal{M}, s_0 \vDash a$ iff $s_0[\![a]\!]s_1$, where $a \in A$, and s_1 is next state of s in S.
3. $\mathcal{M}, s_0 \vDash \neg\phi$ iff $\mathcal{M}, s_0 \nvDash \phi$.
4. $\mathcal{M}, s_0 \vDash \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s_0 \vDash \phi_1$ and $\mathcal{M}, s_0 \vDash \phi_2$.
5. $\mathcal{M}, s_0 \vDash \phi_1 \rightarrow \phi_2$ iff $\mathcal{M}, s_0 \nvDash \phi_1$ or $\mathcal{M}, s_0 \vDash \phi_2$.
6. $\mathcal{M}, s_0 \vDash \forall x : \phi$ iff for all a, $\mathcal{M}, s_0 \vDash \phi(x/a)$.
7. $\mathcal{M}, s_0 \vDash \exists x : \phi$ iff for some a, $\mathcal{M}, s_0 \vDash \phi(x/a)$.
8. $\mathcal{M}, s_0 \vDash \Box\phi$ iff $\forall n \geq 0 : \mathcal{M}, s_n \vDash \phi$.
9. $\mathcal{M}, s_0 \vDash \Diamond\phi$ iff $\exists n \geq 0 : \mathcal{M}, s_n \vDash \phi$.
10. $\mathcal{M}, s_0 \vDash \bigcirc\phi$ iff $\mathcal{M}, s_1 \vDash \phi$.
11. $\mathcal{M}, s_0 \vDash \phi_1\mathcal{U}\phi_2$ iff $\exists i \geq 0 : \mathcal{M}, s_i \vDash \phi_2 \wedge (0 \leq j < i \rightarrow \mathcal{M}, s_j \vDash \phi_1)$
12. $\mathcal{M}, s_0 \vDash \blacksquare\phi$ iff $\forall n < 0 : \mathcal{M}, s_n \vDash \phi$.
13. $\mathcal{M}, s_0 \vDash \blacklozenge\phi$ iff $\exists n < 0 : \mathcal{M}, s_n \vDash \phi$.
14. $\mathcal{M}, s_0 \vDash \ominus\phi$ iff $\mathcal{M}, s_{-1} \vDash \phi$.
15. $\mathcal{M}, s_0 \vDash \phi_1\mathcal{S}\phi_2$ iff $\exists i < 0 : \mathcal{M}, s_i \vDash \phi_2 \wedge (i < j \leq 0 \rightarrow \mathcal{M}, s_j \vDash \phi_1)$

6. SPECIFICATION OF UCON POLICIES

In this section we specify the core authorization models in UCON with the temporal logic introduced before. In an authorization model, usage control policies are specified in logic formulas, which have to be satisfied by the logic model. In these logic formulas, actions are predefined, and predicates are built from subject and object attributes.

6.1 The Model $preA_0$

As presented in [18, 19], most traditional access control models can be expressed in $preA_0$ model, in which an authorization decision is determined by the system before the access happens, and there is no update for subject or object attributes. The usage control policies have the following form:

$$p_1 \wedge ... \wedge p_i \rightarrow permit(s, o, r)$$
$$tryaccess(s, o, r) \wedge permit(s, o, r) \rightarrow \bigcirc\big(permitaccess(s, o, r)\big)$$

where p_1, ..., p_i are state predicates. The $permit$ predicate states that s can access o with r. The $permitaccess$ action grants the permission to s and starts the access. Since there is no update between the access request and grant action, $permitaccess$ will happen in the "next" state of $tryaccess$.

Example 2 BLP model:
subject attribute: clearance
object attribute: classfication

$$dominate(s.clearance, o.classfication) \rightarrow permit(s, o, read)$$
$$tryaccess(s, o, read) \wedge permit(s, o, read) \rightarrow \bigcirc\big(permitaccess(s, o, read)\big)$$
$$dominate(o.classfication, s.clearance) \rightarrow permit(s, o, write)$$
$$tryaccess(s, o, write) \wedge permit(s, o, write) \rightarrow \bigcirc\big(permitaccess(s, o, write)\big)$$

where $dominate$ is a binary predicate of subject attribute and object attribute.

Example 3 Discretionary access control (DAC) model with access control list (ACL):
subject attribute: ID
object attribute: $acl = \{(ID, r)\}$, where ID is a subject identification attribute, r is an access method with which this subject can access this object.

$$in((s.ID, r), o.acl) \rightarrow permit(s, o, r)$$
$$tryaccess(s, o, r) \wedge permit(s, o, r) \rightarrow \bigcirc\big(permitaccess(s, o, r)\big)$$

6.2 The Model $preA_1$

In $preA_1$, authorization rules are checked before the access, and there are one or more update actions before the system grants permission to a subject. The usage control policies are:

$$p_1 \wedge ... \wedge p_i \rightarrow permit(s, o, r)$$
$$permitaccess(s, o, r) \rightarrow \blacklozenge\big(tryaccess(s, o, r) \wedge permit(s, o, r) \wedge \Diamond(preupdate(attribute))\big)$$

where $attribute$ is either a subject or an object attribute. The first rule is the same as before. The second rule says that when a $permitaccess$ happens, an access request must have happened before, the $permit$ predicate had been $true$, and there was a $preupdate$ action happened after that. In this formula, the $permit$ predicate is only required to be $true$ before the action $preupdate$. Note that we assume the time series is bounded during a life cycle of an access period. The "Once" operator does not refer to any past actions before $tryaccess$. The same assumption is also made for future temporal operators.

Example 4 DRM pay-per-use application:
subject: Alice, with attribute $credit$
object: ebook1, with attribute $value$
right: read

$$dominate(Alice.credit, ebook1.value) \rightarrow permit(Alice, ebook1, read)$$
$$permitaccess(Alice, ebook1, read) \rightarrow \blacklozenge\big(tryaccess(Alice, ebook1, read) \wedge \Diamond(preupdate(Alice.credit))\big) \wedge permit(Alice, ebook1, read)$$
$$preupdate : Alice.credit' = Alice.credit - ebook1.value$$

The first rule specifies that whenever Alice's credit is more than the value of ebook1, she can read it. The second rules says that the granting of permission to Alice implies an update of her credit. The $preupdate$ results in a new value of Alice's credit by subtracting ebook1's value from the original credit.

6.3 The Model $preA_3$

In $preA_3$, authorization rules are checked before the access, and there are one or more update actions after the usage process. The usage control policies are:

$$p_1 \wedge ... \wedge p_i \rightarrow permit(s, o, r)$$
$$permitaccess(s, o, r) \rightarrow \blacklozenge(tryaccess(s, o, r)) \wedge permit(s, o, r)$$
$$endaccess(s, o, r) \rightarrow \Diamond(postupdate(attribute))$$

The first two rules are the same as before, except that the update action does not appear in the second rules. The third rules says that a $postupdate$ action will be performed by the system after an access is ended by a subject. Since authorization rules are not enforced after granting access, there is no access revoke in this model.

Example 5 DRM membership-based application:
subject: Alice, with attributes of ID and total $expense$
object: book1, with attributes of $title$ and $readingCost$
subject: readingGroup, with attribute $readerList = \{ID1, ID2, ...\}$ and $bookList = \{book1.title, book2.title, ...\}$
right:read

$$in(Alice, readingGroup.readerList) \wedge in(book1.title, readingGroup.bookList) \rightarrow permit(Alice, book1, read)$$
$$permitaccess(Alice, book1, read) \rightarrow \blacklozenge(tryaccess(Alice, book1, read)) \wedge permit(Alice, book1, read)$$
$$endaccess(Alice, book1, read) \rightarrow \Diamond(postupdate(Alice.expense))$$
$$postupdate : Alice.expense' = Alice.expense + ebook1.readingCost$$

In this example, the authorization policy states that if both Alice and book1 belong to $readingGroup$, she can read the book. Alice's expense is updated by adding the cost of this book after the access.

6.4 The Model onA_0

In pre-authorization models, there is no security check after a system grants a permission. In onA_0, authorization policies are enforced during the access period. The usage control policy is given below:

$$permitaccess(s, o, r) \rightarrow \Box\big(\neg(p_1 \wedge ... \wedge p_i) \wedge (state(s, o, r) = accessing) \rightarrow revokeaccess(s, o, r)\big)$$

In the onA_0 model, authorization predicates have to be satisfied during the access period, otherwise the access will be revoked by the system immediately. The formula says that, after the action $permitaccess$, the formula
$\neg(p_1 \wedge ... \wedge p_i) \wedge (state(s, o, r) = accessing) \rightarrow revokeaccess(s, o, r)$
must be always true: that is, either the authorization predicates $p_1, ..., p_i$ are true when the subject is $accessing$ the object, or the access will be revoked in the next step.

Since we are talking about the core aspects of UCON, in ongoing-authorization models we do not include the pre-authorization rules. In practice, real applications may be expressed by more than one core model.

Example 6 In an organization, a user Bob (with role $employee$) has a temporary position to conduct a short-term project with a certificate of $temp\_cert$. When Bob is accessing some sensitive information, his digital certificate ($temp\_cert$) for this project is being checked repeatedly. If his certificate is in the Certification Revocation List (CRL) of the organization, his temporary role membership is revoked and he cannot access any object any more. The control rule for the access is:

$$permitaccess(Bob, o, r) \rightarrow \Box\big(\neg((Bob.role = employee) \wedge (Bob.temp\_cert \in RCL)) \wedge (state(Bob, o, r) = accessing) \rightarrow revokeaccess(Bob, o, r)\big)$$

6.5 The Model onA_1

In onA_1, there are one or more update actions before a subject starts to access an object. The control rules are:

$$permitaccess(s, o, r) \rightarrow \blacklozenge\big(tryaccess(s, o, r) \wedge \Diamond(preupdate(attribute))\big)$$
$$permitaccess(s, o, r) \rightarrow \Box\big(\neg(p_1 \wedge ... \wedge p_i) \wedge (state(s, o, r) = accessing) \rightarrow revokeaccess(s, o, r)\big)$$

As we have mentioned, there is no authorization check before a subject starts to access an object. So in the first rule, the $permitaccess$ action implies only an update action before it.

6.6 The Model onA_2

In onA_2, there are one or more update action(s) during a usage period. The control policies are:

$$permitaccess(s, o, r) \rightarrow \Box\big(\neg(p_1 \wedge ... \wedge p_i) \wedge (state(s, o, r) = accessing) \rightarrow revokeaccess(s, o, r)\big)$$
$$endaccess(s, o, r) \vee revokeaccess(s, o, r) \rightarrow \blacklozenge\big(permitaccess(s, o, r) \wedge \Diamond(onupdate(attribute))\big)$$

In the second rule, we only specify that there is an update action. In many applications there are cases where an update is always required in every state of the ongoing access. We will illustrate this with an example later.

6.7 The Model onA_3

In onA_3, there must be update action(s) after a usage process. The control policies are:

$permitaccess(s, o, r) \rightarrow \Box\big(\neg(p_1 \wedge ... \wedge p_i) \wedge (state(s, o, r) = accessing) \rightarrow revokeaccess(s, o, r)\big)$
If an access is ended by subject:
$endaccess(s, o, r) \rightarrow \Diamond(postupdate(attribute))$
If an access is revoked by system:
$revokeaccess(s, o, r) \rightarrow \Diamond(postupdate(attribute))$

In many applications, the update after an access is ended by a subject is different from the one after an access is revoked by the system. Here we simply use the same action name of $postupdate$, but they may change the attribute to different values, or change different attributes.

Example 7 Consider the usage control policies in Example 1, in which one object attribute is a set of accessing subjects $accessingS = \{s | s \text{ is accessing } o\}$. For different policies we define different subject attributes.

a . Revocation by the earliest start time:
We define the starting time ($startTime$) as a subject attribute. The authorization rules are:

(1) **true** $\rightarrow permit(s, o, r)$
(2) $permitaccess(s, o, r) \rightarrow$
$\blacklozenge(preupdate(o.accessingS))$, where $preupdate$: $o.accessingS' = o.accessingS + \{s\}$
(3) $tryaccess(x, o, r) \wedge (x \notin o.accessingS) \wedge (|o.accessingS| = 10) \wedge (s \in o.accessingS) \wedge (s.startTime = Min_{startTime}(o.accessingS)) \rightarrow revokeaccess(s, o, r)$
(4) $endaccess(s, o, r) \vee revokeaccess(s, o, r) \rightarrow \Diamond(postUpdate(o.accessingS)$, where $postUpdate$: $o.accessingS' = o.accessingS - \{s\}$

$|o.accessingS|$ is a function returning the number of accessing subjects, and $Min_{startTime}(o.accessingS)$ is a function returning the earliest start time from $accessingS$. The first rule says that a new access request is always permitted. The second rule specifies that whenever a subject tries to access the object, there must be a pre-update of $accessingS$ before the subject starts to access. The third rule says that when the total number of accessing user is 10, and a new request is generated, the subject with earliest start time will be revoked. The fourth rule specifies a post-update needed when the access is ended or revoked (the updates are the same in this example).

b . Revocation by the longest idle time:
We define two subject attributes: the status of the usage ($status$ with value of $busy$ or $idle$) and continuous idle time in a single usage period ($idleTime$). The control rules are:

(1) **true** $\rightarrow permit(s, o, r)$
(2) $permitaccess(s, o, r) \rightarrow$
$\blacklozenge(preupdate(o.accessingS))$, where $preupdate$: $o.accessingS' = o.accessingS + \{s\}$
(3) $tryaccess(x, o, r) \wedge (x \notin o.accessingS) \wedge (|o.accessingS| = 10) \wedge (s \in o.accessingS) \wedge (s.idleTime = Max_{idleTime}(o.accessingS)) \rightarrow revokeaccess(s, o, r)$
(4) $\Box\big((s.state = idle) \rightarrow onupdate(s.idleTime)\big)$
(5) $endaccess(s, o, r) \vee revokeaccess(s, o, r) \rightarrow \Diamond(postupdate(o.accessingS))$, where $postupdate$: $o.accessingS' = o.accessingS - \{s\}$

The $Max_{idleTime}(o.accessingS)$ is a function returning the largest $idleTime$ in $accessingS$. Rules (1), (2), and (5) are the same as before. In rule (3), the revocation is determined by the $s.idleTime$. Rule (4) specifies the mutability of the subject attribute by saying that there must be a continuous update of $s.idleTime$ performed by the system whenever the state of a subject is $idle$.

c . Revocation by the longest total usage time:
We define the accumulating usage time ($usageTime$) as a subject attribute. The control rules are:

(1) **true** $\rightarrow permit(s, o, r)$
(2) $permitaccess(s, o, r) \rightarrow$
$\blacklozenge(preupdate(o.accessingS))$, where $preupdate$: $o.accessingS' = o.accessingS + \{s\}$
(3) $tryaccess(x, o, r) \wedge (x \notin o.accessingS) \wedge (|o.accessingS| = 10) \wedge (s \in o.accessingS) \wedge (s.usageTime = Max_{usageTime}(o.accessingS)) \rightarrow revokeaccess(s, o, r)$
(4) $endaccess(s, o, r) \vee revokeaccess(s, o, r) \rightarrow \Diamond(postupdate(usageTime) \wedge \Diamond(postupdate(accesingS))$, where $postupdate$: $o.accessingS' = o.accessingS - \{s\}$

The $Max_{usageTime}(o.accessingS)$ is a function returning the largest $usageTime$ in $accessingS$. Rules (1), (2), and (3) are the same as those in b, while rule (3) specifies that the revocation is determined by the total usage time of the subject. Rule (4) says that after each usage, there must be an update on $usageTime$.

7. OBLIGATIONS AND CONDITIONS

Obligations and conditions are two other important components in usage decision of UCON besides authorizations. In this section we discuss the logical approach to these two aspects.

7.1 Obligations

There are different views and philosophies on obligations from the literature. Bettini et al. [6, 7] presented a policy management system on provisions and obligations, where a provision is an action to be performed before, while an obligation is an action that will be performed in the future. Chomicki and Lobo [9] investigated the conflicts and constraints of historical actions in policies. They used past linear-time temporal connectors to express action constraints, such as two actions cannot happen at the same time or in certain sequence. In these papers, obligations are regarded as actions to be performed by the system or a subject. In our model we adopt the same view on obligation in UCON policies. In Fig. 1 we show that obligations form a component of usage decision to control access requests or ongoing usage. Obligations that have to be performed after an access, since they only affect the future usage process, are considered as global obligations [18, 19]. For example, an action of a user clicking an agreement button before playing a music file is regarded as an obligation, while the payment action of monthly billing is a global obligation, because this action doesn't affect the current usage access. In UCON we need an administration model to capture the global obligations. In this paper we only focus on the session-based usage control model, in which only obligations before and during the usage process are considered. The global obligations will be described in future work on administrative models.

Because of the continuity of usage decision, there are two types of obligations in UCON: pre-obligation and ongoing-obligations:

1. $preB$: obligations that must have been performed before a subject starts to access an object.
2. onB: obligations that must be performed during a usage process.

DEFINITION 3. *An obligation action is described by* $a_b(s, o, r, s_b, o_b, r_b, para_1, ..., para_i, ...)$ *where* a_b *is the obligation name,* (s, o, r) *is a particular usage process requiring the obligation,* s_b*,* o_b*,* r_b *are obligation subject, object and right,* $para_1$*, ...,* $para_i$ *are optional parameters that may be needed to specify the action, such as time, amount, etc.*

s_b, o_b, r_b may be the same as that in (s, o, r), or different, depending on particular applications. For example, downloading a music file may need the same subject's obligation of clicking a privacy button, while a child's watching an online movie may need the parent's agreement.

DEFINITION 4. *A logical model of UCON with authorizations and obligations is a 4-tuple:* $\mathcal{M} = (\mathcal{S}, \mathcal{P}, \mathcal{A}_\mathcal{A}, \mathcal{A}_\mathcal{B})$*, where* $\mathcal{S}$ *is a sequence of states,* $\mathcal{P}$ *is a finite set of predicates,* $\mathcal{A}_\mathcal{A}$ *is a finite set of authorization actions (same as the* $\mathcal{A}$ *in the authorization model),* $\mathcal{A}_\mathcal{B}$ *is a finite set of obligation actions.*

Example 8 In an online electronic marketing system, only a registered user can browse all the products. Before placing orders, a customer has to open a terms account, which requires financial information. To place an order, a customer has to click a button to agree to the order policies. The "click" action is an obligation. A role of $registered$ will be assigned to a customer after the registration. We define an action $click\_agreement$, which is an obligation for each order. The control policy is:
$(s.role = registered) \rightarrow permit(s, o, order)$
$permit(s, o.order) \wedge$
$\ominus(click\_agreement(s, o, order, s, agree\_statement, click))$
$\rightarrow permitaccess(s, o, order)$

7.2 Conditions

Conditions are environmental restrictions that have to be valid before or during a usage process. Normally, this environmental information is not directly related to a subject and an object, and thus is not considered as subject or object attributes, but system attributes. Similar to obligations in UCON, we only focus on preconditions and ongoing-conditions. Since post-conditions do not affect the current usage request, we will consider this in an administrative model in future work.

In Section 5 we mentioned that there may be some system attributes besides $state(s, o, r)$. A condition is a state predicate built from one or more system attributes. For example, a subject can only get a permission when the system clock is in daytime, or in a particular period during daytime.

DEFINITION 5. *A logical model of UCON with authorizations, obligations, and conditions is a 5-tuple:*
$\mathcal{M} = (\mathcal{S}, \mathcal{P}_\mathcal{A}, \mathcal{P}_\mathcal{C}, \mathcal{A}_\mathcal{A}, \mathcal{A}_\mathcal{B})$*, where* $\mathcal{S}$*,* $\mathcal{A}_\mathcal{A}$*, and* $\mathcal{A}_\mathcal{B}$ *are the same as before,* $\mathcal{P}_\mathcal{A}$ *is a finite set of authorization predicates (the* $\mathcal{P}$ *before), and* $\mathcal{P}_\mathcal{C}$ *is a finite set of condition predicates.*

Example 9 Suppose that a day-shift user ($dayshifter$) can only access an object during daytime, and a night-shift user ($nightshifter$) can only access the object during nighttime. We define the system time $current\_time$ as an attribute. Note that this is an environment status, not an attribute of any subject or object. The usage control policies are:

$(s.role = dayshifter) \wedge (8am \leq currentT \leq 5pm) \rightarrow permitaccess(s, o, r)$
$(s.role = nightshifter) \wedge \neg(8am \leq currentT \leq 5pm) \rightarrow permitaccess(s, o, r)$

8. RELATED WORK

Bertino et al. [2, 3, 4] introduced a temporal authorization model for database management systems. In their 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 authorization model is different from this. We consider the temporal characteristics in a single usage period with mutable attributes of subject and object before, during and after access, that is, the temporal properties are the results of mutability of subject and object attributes, which change due to the side-effects of accesses and usages. In contrast Bertino et al.'s model focuses on the validity of authorization policies with time period, and the temporal property of a policy is not related to an access action, but dependent on system administration policies. Gal et al. [11] proposed a temporal data authorization model (TDAM) for access control to temporal data. This work is orthogonal to our approach, since we focus on the temporal authorization and usage process, while TDAM focused on the temporal attributes of data. For formal specifications with temporal logic in security policies, Siewe et al. [22] applied interval temporal logic to express and compose access control polices, and Hansen and Sharp [12] introduced an approach for the analysis of security protocols using interval logic. The main difference in our approach is that we use TLA in our logic specification, and we focus on the atomic actions and temporal properties during a single usage process, while their approaches focus on a higher level of system policies or security protocols.

Bettini et al. [6, 7] presented concepts of provisions and obligation in policy management. In their papers, provisions are conditions or actions performed by a subject before the authorization decision, while obligations are conditions or actions performed after an access. In our model, we distinguish 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] investigated the conflicts and constraints of historical actions in policies. In their paper, actions are application activities, and constraints are expressed with linear-time temporal connectors. In our paper we define obligations as actions required by an access, and represent the logic approach with TLA.

9. CONCLUSIONS AND FUTURE WORK

We have developed a logic specification of UCON with temporal logic of actions. A logic model is presented with a sequence of states including subject, object, and the system. The authorization predicates are built from the subject and object attributes, while actions are the state transitions of subjects, objects, and the system. Temporal formulas are built from authorization predicates and actions, which hold in a model as usage control policies. We also discussed our logic approach to obligations and conditions. An obligation is an action that has to be performed before or during an access, while a condition is a predicate with system attributes. The powerful specification capability and flexibility of TLA strengthens UCON with precise modelling and specification of security policies.

There are several ongoing and future directions for this work. Specifically, we will continue the work on the following aspects:

1. Enrich the model of UCON. The original UCON is a high-level model, and only core concepts are presented. We are going to enrich the model with more details, such as the right attributes, as well as mutual constraints, like dynamic separation of duty and history based constraints.

2. A UCON system needs an administrative model, which manages the attributes, administration policies, etc. UCON is attribute-based, which requires synchronized attribute acquisition and management. Also, post-obligations and conditions are in the scope of the administrative model. If a subject does not satisfy any obligation after an access, a security administrator needs to take some compensatory actions according to the administrator policies.

3. Safety is an important and basic problem in access control systems. In UCON, the safety problem is to determine if a subject can get a particular permission to an object with his/her current attributes, as well as any future updates of these attributes by performing some accesses.

4. As mentioned in Section 1, concurrency is a unique feature in UCON, which has been seldom investigated in access control. In an open system, an update action of an attribute will result in the authorization decision change in another access, which is happening concurrently, or is going to happen in the future. We will apply some tools such as TLA+ [16] to specify access control in such open and concurrent environments.

10. REFERENCES

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

[2] E. Bertino, C. Bettini, and P. Samarati, A Temporal Authorization Model, In Proc. of ACM Computer and Communication Security, 1994.

[3] E. Bertino, C. Bettini, E. Ferrari, and P.Samarati, A Temporal Access Control Mechanism for Database Systems, IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.1, Feburary 1996.

[4] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning, ACM Transaction on Database Systems, Vol.23, No.3, September 1998.

[5] 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.

[6] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera, Obligation Monitoring in Policy Management, 3rd Internationl Workshop on Policies for Distributed Systems and Networks, 2002.

[7] C. Bettini, S. Jajodia, X. Sean Wang, and D. Wijesekera, Provisions and Obligations in Policy Management and Security Applications, In proc. of the 28th VLDB Conference, Hong Kong, China, 2002.

[8] D. Brewer and M. Nash, The Chinese Wall Security Policy, IEEE Symposium On Research in Security and Privacy, Oakland, California, 1988.

[9] J. Chomicki and J. Lobo, Monitors for History-Based Policies, 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] A. Gal and V. Atluri, An Authorization Model for Temporal Data, In Proc. of ACM Computer and Communication Security, 2000.

[12] M. Hansen and R. Sharp, Using Interval Logics for Temporal Analysis of Security Protocols, In Proc. of the ACM Workshop on Formal Methods in Security Engineering, 2003.

[13] S. Jajodia, P. Samarati, and V. S. Subrahmanian, A Logical Language for Expressing Authorizations, IEEE Symposium On Research in Security and Privacy, Oakland, California, 1997.

[14] 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.

[15] Leslie Lamport, The Temporal Logic of Actions, ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 16 Issue 3, May 1994 (also available from http://research.microsoft.com/users/lamport/tla/papers.html).

[16] Leslie Lamport, Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2003.

[17] Zohar Manna and Amir Pnueli, The Temporal Logic of Reactive and Concurrent Systems Specification, Springer-Verlag, 1991.

[18] 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.

[19] J. Park and R. Sandhu, The $UCON_{ABC}$ Usage Control Model, ACM Transactions on Information and Systems Security, Feb., 2004.

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

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

[22] F. Siewe, A. Cau, and H. Zedan, Compositional Framework for Access Control Policies Enforcement, In Proc. of the ACM Workshop on Formal Methods in Security Engineering, 2003.