World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:3, 2014

Component Lifecycle and Concurrency Model in Usage Control (UCON) System

P. Ghann, J. Shiguang, C. Zhou

***Abstract*—**Access control is one of the most challenging issues facing information security. Access control is defined as, the ability to permit or deny access to a particular computational resource or digital information by an unauthorized user or subject. The concept of usage control (UCON) has been introduced as a unified approach to capture a number of extensions for access control models and systems. In UCON, an access decision is determined by three factors: authorizations, obligations and conditions. Attribute mutability and decision continuity are two distinct characteristics introduced by UCON for the first time. An observation of UCON components indicates that, the components are predefined and static. In this paper, we propose a new and flexible model of usage control for the creation and elimination of some of these components; for example new objects, subjects, attributes and integrate these with the original UCON model. We also propose a model for concurrent usage scenarios in UCON.

***Keywords*—**Access Control, Concurrency, Digital container, Usage control.

I. Introduction

USAGE control has been introduced as a comprehensive access control model in a highly distributed network-connected environment. Compared to traditional access control models and their derivatives, where access decision is based on only authorization, the concept of usage control (UCON) has been introduced as an amalgamated approach to capture a number of extensions for access control models and systems. In UCON, a control decision is determined by three significant factors: authorizations, obligations and conditions. Additionally usage control also introduces attribute mutability and decision continuity as two distinct and nascent characteristics. The concept of usage control helps to address current information technology needs especially in electronic commerce: by providing additional features necessary for access control. This provides a fine-grained control that helps to achieve integrity, confidentiality and availability of information or resource. Usage control decision can be enforced by policies of authorization, obligation as well as condition. Decision can also be enforced before (pre), during (on) or after (post) access. In the process of enforcing usage decision, actions may lead to change in the state of the system or even status of access. These changes must conform to policies of access hence access will be revoked. In other words a granted access may be revoked by the system if certain policies are not satisfied, based on changes in the subject or object attributes, or environmental conditions, or some obligations that are not fulfilled during the usage process. Though UCON is a comprehensive concept compared to traditional access control, it has been observed that most of it components are predefined and static. In view of this our contribution in this paper is to enhance on the expressiveness of UCON by looking at the life cycle of the various components of UCON such as subject, object and attributes. We also look at concurrent usage scenarios of UCON since the original UCON model considers single usage scenario. The rest of this paper is presented as follows: Section II is on usage control and its components. In Section III, we propose a lifecycle for the some of the components of UCON. Section IV is on the storage of objects in digital container. We come out with a model for concurrent implementation of usage control using authorization core models in Section V. Section V *A* is related work. And we conclude this paper in Section VI.

Patricia Ghann is with the School of Computer Science and Telecommunication Engineering, Jiangsu University, Jiangsu, China (corresponding author, phone: +8618306105093, e-mail: pghann@gmail.com)

Ju Shiguang is with the School of Computer Science and Telecommunication Engineering, Jiangsu University, Jiangsu, China (e-mail: *jushig@ujs.edu.cn*)

Conghua Zhou is with the School of Computer Science and Telecommunication Engineering, Jiangsu University,Jiangsu, China (phone: +8613656138071, e-mail: chzhou@ujs.edu.cn)

II. Usage Control (UCON) and Its Components

UCON model as mentioned earlier, addresses access control challenges in modern application and computing environments. Access permission in UCON is based on attributes and three main decision factors, authorization, obligation and condition. Significantly UCON enhances upon traditional access control in two main aspects; mutability of attributes and continuity of access decisions. The UCON model consist of eight core components as shown in Fig. 1; subjects, subject attributes, objects, object attributes, rights and three main decision factors; authorization, obligation and condition [1]. Subjects and objects are similar concept with traditional access control models. Subject and object attributes are used during the decision process. Subject attributes include identities, group names, roles, membership, security clearance etc. Objects are entities that subjects hold rights on and can therefore access the object. Attributes of objects are properties of the object that is used in decision process. These include security labels, class, price, ownership etc. [2].

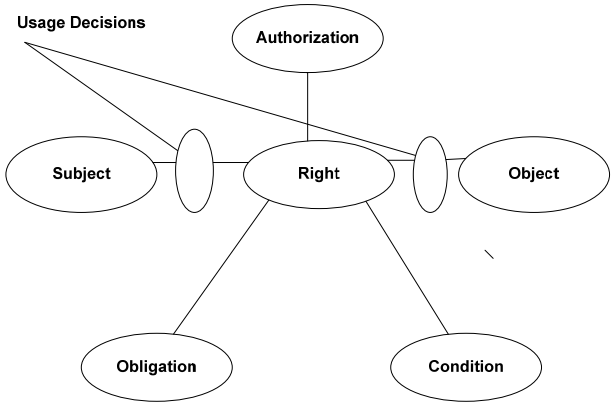

Fig. 1 Components of Usage Control

III. UCON Components' Lifecycle

A. Subject and Subject Attributes

Subjects are entities that hold right on objects. In UCON, a subject specifically refers to a user or human being and considers mainly consumer subject. In our view, consumer subjects must possess a unique attribute that transcends all security domains. This is especially important in ecommerce sector. The use of attributes such as username, passwords, emails and the provision of answers to certain system-generated questions by users or subjects, are easily transferred, and most of the time not trustworthy and must be avoided in order to ensure security of resources or digital information. For example, a subject can have more than one username, email, and in terms of providing answer to system-generated questions, might not be who he or she claims to be. Additionally UCON system must specify policies that state the duration of access by every subject. For instance, a year of access can be granted to a user based on the particular type of resource or object. Thus, a policy can specify the life span for an object or resource to reside in the UCON system as well as the duration of access permitted to users; a particular resource would reside or be available in the system for a stipulated time. For instance, with regards to sensitive resources such as bank statements and medical reports, a policy can be specified that allow owners of these resources a maximum of three times access for one year availability of resource in the system, after which access is denied. After the stipulated life span of the resource has expired, the system is expected to deny all attempts to access a resource and also eliminate it from the system. With this, subjects would have three times authorized access to such resources within year. Additionally constraints can be specified using factors such as obligation and condition to enforce security. According [3], [4] time authorization is given by:

tuple (*pt, s, o, priv, pn, g*), where $pt \in N$; $s, g \in S$; $o \in O$; $priv \in P$;

pn $\in \{\}$; *tuple* (*pt, s, o, priv, pn, g*) states that users have been authorized (*if pn = '+'*) or denied (*if pn* = '-') for *'pt*' times *'privilege (priv)'* on object *o* by user *g*.

For instance, *tuple* (*6, Tom, Sun, read, +, Sam*) denotes that Sam authorizes 6 times privilege *read* on the book *Sun* to Tom. To express the idea of assigning life span to objects, which automatically affects the right of access to such an object, we modified the time authorization formula as:

(*D, S, type* (*o*)*, T* (*right*)*, pn, UCONs*)

where *D* is a natural number representing the duration of access which can be in days, weeks, months or years. *S* represent subject, type(*o*) represent the type of object such as sensitive, intellectual and non-sensitive objects, T(right) represent the number of Times a right can be executed, pn represent +, - and in this case stands for permitaccess or denyaccess after a subject has executed the tryaccess action and UCON**s** represent the UCON system. Based on our modification, our previous time authorization example can be rewritten as; (6M, Tom, Sensitive Object, 2Read, +, UCON**s**). What this simply implies is the UCON system authorizes the reading of a sensitive object by the subject Tom, twice a month for 6 months. It also communicates to the subject the availability of such an object in the UCON system and hence the lifespan of his or her access right.

Hypothesis:

A time authorization is given by a six tuple as:

(pt, s, o, priv, pn, g), where pt $\in$ N; s, g $\in$ S; o $\in$ O; priv $\in$ P; pn $\in \{\}$.

tuple (pt, s, o, priv, pn, g), specify that a user has been authorized (if pn = '+') or denied (if pn = '-') for pt times priv on object o by user g.

We modify this formula to capture the life span of object or resource as;

(*D, S, type* (o), *T*(*right*)*, pn, UCONs*),

where *D* is a number, type (*o*) represents the type of object, *T* (*right*) represents the number of times a right can be executed, *pn* = {} where after tryaccess action, + is permitaccess, and – is denyaccess, UCONs is the UCON system.

B. Object and Object Attributes

Objects are entities that subjects have rights on. Thus subjects can access or use objects when granted the permission to do so. Objects must therefore be classified and stored appropriately in a digital container based on their type. For instance sensitive object or resource must be stored in a container different from non-sensitive and intellectual resources. This would ensure the ease with which life span is assigned to particular group of resource and also monitor access to such resource by users in a particular container. In [5] a formal mode is proposed that specifies how to create and destroy objects, subjects and their attributes in a UCON system by considering a serialized usage processes. The formal model however focuses on pre-authorization and pre- obligation with post-update. Thus the model does not focus on ongoing usage scenarios but considers only the overall effect induced by a

sequence of non-interfering usage processes in creating or destroying new subjects or objects and updating attributes after wards. The usage control policy in their model consists of two parts - a *condition* and *body*:

policy_name(s, o); (condition): $p_1, p_2, \ldots, p_n \rightarrow$ permit(s, o, r);
(body): $act_1, act_2, \ldots act_k$

The *condition* part contains access rules. The access rules are represented as a conjunction of authorization, obligation and condition predicates. If the conjunction is true, the access rule grants a certain access right to a subject. The second part of the policy, *body*, is a sequence of primitive actions. If the conditional part of the usage control policy is satisfied, the UCON protection system enforces the second part and moves the system to the new state executing a sequence of primitive actions. Not only is this formal model deficient due to its inability to consider ongoing usage processes, it fails to state under what circumstances objects or subjects must be created and how long created objects must be contained in a digital container before they are destroyed. From our point of view, object, should be created by object provider or the subject who owns this object. The destruction of these objects however can be done by the UCON system or resource provider in accordance with specified policies of the UCON system. Thus the creation or destruction of resources and their attributes must conform to policies that pertain to the UCON system and also to an organization. As previously stated, sensitive object and as such all other types of objects or resources created by a resource provider or resource owner, need to be assigned a life span by which it must be contained in a digital container within the UCON system. When the life span is due, the system must automatically destroy the object and deny access to all users. In other words, different containers are expected to have different life span since their contents inherently differ as illustrated in Fig. 2. Agreeably, a container with sensitive resources or objects such as bank statements, medical report would undeniably have a short life span; to guarantee security and privacy. On the other, most intellectual resources such as eBooks, music, movies are in themselves renewable. For instance an eBook tag as first edition unquestionably cannot be stored in the same container with its second or latest edition. The same applies to current as well as old movies and music. A digital container with these type of resources ought to be indisputably assigned a specific life span such that, old versions of resources are eliminated from the container by the system and new ones created appropriately by resource or service provider. By assigning life span to digital containers, old resources can be easily eliminated and new ones created when the life span is due or expires. Also the right of access to resources contained in these digital containers by users would have to be eliminated and new right of access created for new resources. The enforcement of this would help enhance the performance of the UCON system and also prevent rights and resources from abuse.

C. Rights

Rights in UCON are the same as access rights and permission in traditional access control [6]. In UCON, rights imply usage permissions which a subject can utilize on objects. Furthermore the existence of a particular right is not predefined, but rather determined when an access is attempted by a subject depending on the subject, object and environmental attributes, as well as authorization, obligation and condition. Thus rights exist only when there is an object or resource in a system and an attempt is made by a subject to access this object or resource. Rights are permitted when access constraints are satisfied and denied otherwise. For instance in the process of usage if updates occur and these updates do not conform to policies of access, the granted right is revoked by the system.

D. Authorization

Authorization is functional predicate over attributes of subject and object that must be satisfied in order to grant access to a particular resource. In UCON model, this authorization predicate can be evaluated before or during access. For example a policy might specify that a subject must be 18 years to access an object or resource. Thus authorization is stated considering just the subject and object attribute without considering the system in general [7]. From our point of view, the state of the system is very significant where the life span of resources is considered. The system must also be able to determine when to grant access to subject. For instance, the system might be functioning alright; however the object or resource that a subject wants to access might have expired or the subject right of access has reached its limit. In which case, a subject needs not try access since the system would automatically deny access.

E. Obligation

The UCON system mainly deals with the fulfillment of obligation before, during or after the usage processes. Obligation in UCON is a tuple of OBL = (OBS, OBO, OBA, WHEN, DURATION) during the time of access; where OBL is the obligation element, OBS and OBO are the obligation subject and obligation object respectively, OBA is the obligation action, WHEN represent |pre |on |post obligation action that must be performed and Duration is the time frame within which obligation has to be fulfilled [8], [9]. UCON system does not consider obligations that have to be fulfilled after object or resource have been rightfully accessed. In our previous paper we looked at the concept of enforcing usage control on a remote client server and proposed ways by which post obligations can be fulfilled. Obligations as decision factor will be of no importance without a consideration of the type of object or resource. For instance obligations that pertain to sensitive objects would require different mechanism to enforce and monitor their fulfillment than obligations that pertain to non sensitive resources.

F. Conditions

Conditions are environmental constrains that are taken into account in the usage decision process. Conditions are not

directly related to subjects and objects, but are based on environmental attributes. The appraisal of condition predicates can be done before granting the access to a digital object (*pre*-conditions) and/or when a subject is accessing an object (*on*-conditions). Conditions checks however has no influence on subject, object or environmental attributes. Nonetheless, the value of the conditional status can be changed inherently as the result of environment modifications [10], [11].

IV. Storage and Access to Objects in Digital Container

Digital containers are cryptographic carriers of digital information that utilizes encryption, digital signature and digital certificate to achieve confidentiality and integrity. This mechanism is used by UCON to prevent unauthorized access to protected digital content. Based on this, these containers can also be assigned life span. When the life span assigned to a digital container expires, the system must block and prevent the (tryaccess) action by a subject, and thus deny access automatically.

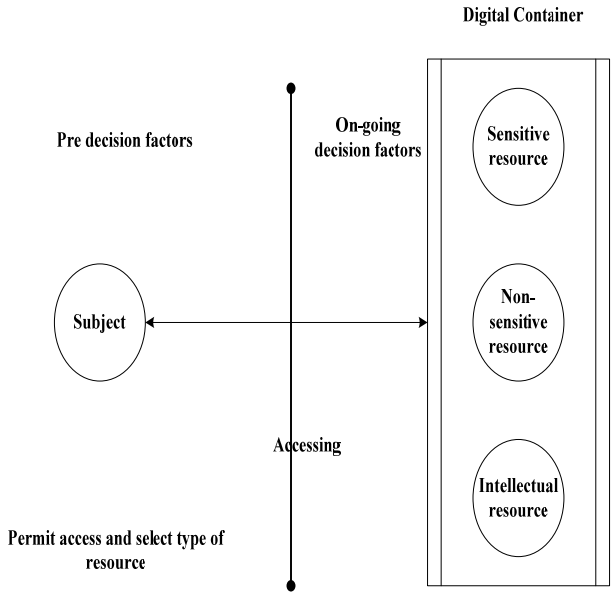

Fig. 2 Access to an object in a digital container

In Fig. 2, after the tryaccess action, the subject is permitted access by selecting the type of object and the type of access. This is then followed by the necessary pre and ongoing actions that have to be performed by the subject or system. Thus access to an object in a digital container depends on the life span of the object in the digital container as well as the number of access by a subject. For an example, let consider an instance where the life span of a resource in digital container is one year and the number of access by a subject is three times. In the above instance, once the system permits access to a subject, the information pertaining to the use of such object is made available to the subject; life span of object and the number of times a subject is allowed to access a particular object. In this case using our hypothesis we can deduce:

Access = (D, S, type (o), T(right), pn, UCONs)

V. Related Work

Resource management has always been the central focus of concern in the concurrent programming [12]. This is because most often than not, a number of processes share access to system resources for example; memory, processor time or network bandwidth. The correct usage and management of resource is of great significance for the overall performance of computational systems [13]. References [14] and [15] looked at the problem of resource control in their work on concurrent programming. Their work emphasized the importance of resource separation as a means of controlling complexity of process interactions and reducing the possibility of dependent errors and also the use of synchronization mechanisms to provide protection from inconsistent usage. A number of approaches to reasoning about imperative concurrent programs have been proposed. However, it is the ideas in an early paper by [16] on concurrency; "Towards a Theory of Parallel Programming (TTPP) that fit well with the view point of separation.

The approach by [16] centered on a concept of "spatial separation"; a way to think about concurrent processes as well as to simplify reasoning. Hoare described formal proof rules for shared-variables concurrency that was modular based on compiler-enforceable syntactic constraints for ensuring separation. Due to the modularity, one could reason locally about a process, because simple syntactic checks ensured that no other process could tamper with its state in a way that invalidated the local reasoning.

In their work [17], inserted the separating conjunction in appropriate places in the TTPP proof rules; thus the extension of the rules studied by [18]. This insertion resulted in two surprises in the proof rules: one positive and another negative. The negative surprise was a counterexample as a result of [19] John Reynolds, who showed that the rules were unsound when used without restriction. The difficulties in showing the soundness delayed the publication of proof rules. Nonetheless after the counterexamples were detected, [20], [21] resolved a similar issue in sequential programming language by requiring certain assertion in the proof to be "precise"; these assertions are those that unambiguously picked out an area of storage.

The positive surprise on the other hand was that they could handle a number of daring and also valuable programming idioms, which as a result opened up a number of unexpected possibilities. According to [22], the idioms involved the transfer of ownership of or right of access to a piece of state from one process to another, which is a common behavior in the system programs, where limited resources are shared among a collection of competing or cooperating processes. The least expected result however was that, their method turned out not to be dependent essentially on having structured notations encapsulating high level uses of synchronization constructs; because they were able to reason in modular way about some semaphore idioms. Consequently they used what they termed as resource reading of semaphores; where a semaphore is (logically) attached to a piece of state and where pieces of state

flow dynamically between the semaphores and surrounding code when P and V operations are performed. According to [23] the ability to deal with ownership transfer is as a result of using a logical connective and separation conjunction to describe a resource partition that change over time.

A program is said to be racy if two concurrent processes attempt to access the some portion of state at the same time: definition 4. This statement is dependent not only on the ordering of potential interleavings but also on the level of granularity of the operation. Thus the avoidance of race is very crucial when processes compete for system resource when resources are to be used to ensure consistency. Races can also lead to irreproducible program behavior, which makes testing difficult. In other words race-freedom frees avoids thinking about tiny details of interleaving or granularity of sequential programming constructs. Sequentially equivalent programs that can be distinguished by concurrency are:

$$X: = X+1,\ X: X+1 \text{ and } X: = X+2$$

The only way to identify their inequivalence is through interference from another process by racing. Though it is not impossible in principle to describe the minute details of interleavings, nonetheless the aim of every program design is to ensure the avoidance of thinking about such minute details, which is basically the point of Dijkstra's criterion of speed independence in his work; principles for concurrent program design. In monitor-based concurrency, each monitor determines a mutual exclusions group, consisting of all cells the monitor produces. In other words when programming with semaphores each semaphores s of condition critical regions with B do C with common resource name r forms a mutual exclusion group. Our work uses the ideas of [13] and [16] of speed independence because UCON is a comprehensive model, consisting of 16 core models with usage decision continuity and mutability of attributes. In other words attribute change while in the process of access and this also results in update actions. In TTPP proof rules, the simplicity and modularity is achieved by syntactic a restriction, which ensures caution. We use two controlling agents to ensure caution and achieve consistency. In contract, [24] analyze an interplay between an interleaving semantics based on traces of actions and a "local enabling" relation that "executes" a trace in a portion of state by a process. We introduce our model in Section V *A*.

VI. Concurrency in UCON

Concurrency is the study of how multiple, independently controlled threads behave when running together and interacting with each other. It has been at the center of development in the computing industry since its inception, and continues to this day. Why is concurrency so important? The answer is simple: in the real world, there's a whole lot going on, and it's all happening concurrently. To build a computer application that effectively deals with concurrency in the real world, one must go to extraordinary lengths to avoid destroying the concurrency required to manage it all and hence ensure ACID in a particular system. Various concurrency control techniques and methods have been proposed to help achieve this. These include optimistic, pessimistic and semi-optimistic. The objective of concurrency systems is to execute multiple threads that share common resource without reducing the system performance or efficiency and at the same time prevent deadlocks [25], [26]. For a long time in the history of computers, locks have been deployed as the best solution for concurrency. We propose a model for concurrency in usage control systems. This is because the original concept of UCON expresses a single usage scenario. In other words, it does not express interactions that occur during concurrent usage sessions. We exemplify a concurrent usage scenario using $UCON_A$. Usage control authorization decision is defined by the value of subject and objects attributes. There are also three types of actions that can influence usage decisions as a result of attribute mutability: preupdate, onupdate and postupdate. These actions can either be performed by the system or subject before access, during or after access; leading to changes in system states [27], [28].

Using the UCON transition states in Figs. 3 (a) and (b), we identify a bone of contention or a racy condition in states S_1 and S_3 respectively after access has been granted. Therefore S_1 and S_3 are regarded as the critical sections of our system. To achieve ACID in each state, our model introduces two different decision control agent; Pre- Decision Control (PDC) and On-Decision Control (ODC) respectively. These control agents are responsible for providing a race free process during concurrent usage sessions and to promote ACID in each state of the system. We illustrate how the proposed model works by considering two processes N_1 and N_2 where N_1 and N_2 are two distinct processes by two subjects trying to use the iTunes system with UCON preA policy.

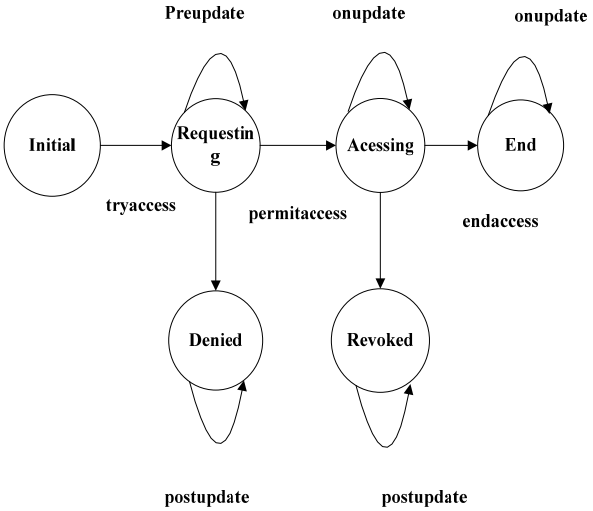

Fig. 3 (a) Transition states of UCON

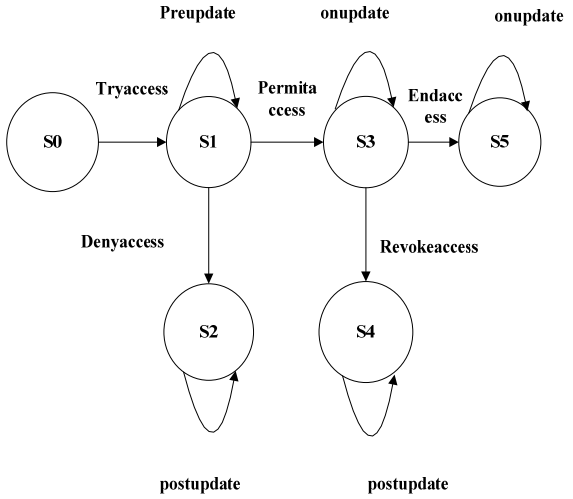

Fig. 3 (b) State transition graph of UCON system

Using the transition states in Fig. 3 (a) and the state transition graph in Fig. 3 (b), the models in Figs. 4 (b) and (c) are implemented for concurrent usage session.

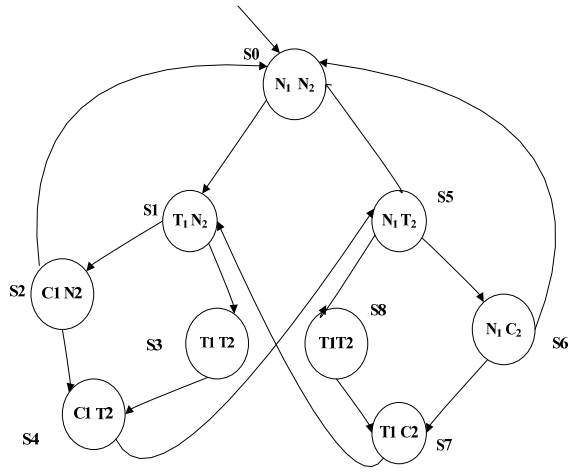

Fig. 4 (a) Model for mutual exclusion

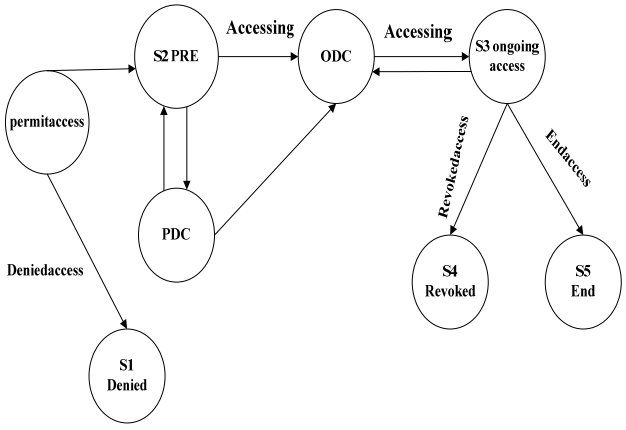

Fig. 4 (b) Usage Control Concurrency Model

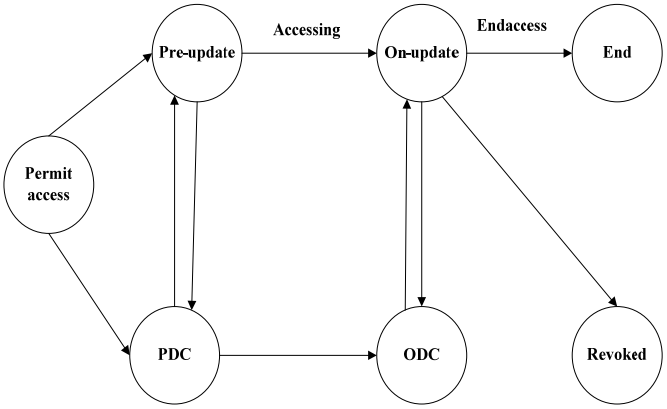

Fig. 4 (c) Usage Control Concurrency Model with permit access

The objective of the proposed model is to help ensure safety, non- blocking, and liveness when two processes are interacting at the same time. In Fig. 4 (b) once access is granted, a signal is sent to the PDC agent informing it of the type of resource or object, the type of access and the type of update; pre or on-going update. We illustrate this with the iTunes system. When processes N_1 and N_2 are both permitted access, both of them proceed without any lock or one process waiting for the other to exit the critical section. This is possible because the PDC sends a signal to the critical section. Based on the information sent, the pre decision state (critical section) performs the necessary preupdates required on copies of the original object with a time stamp.

Using the iTunes system as a case study, both N_1 and N_2 are allowed into the critical section without locking each other. Thus the original object remains in the system without any updates. The timestamp is deployed in order to eliminate updated objects with the longest time from the system. When an on-going authorization is required while accessing platform authorization, both processes; N_1 and N_2 move to the onupdate state (critical section), information is exchanged between the onupdate state and ODC agent for the necessary update to be enforced. Otherwise access is revoked and a postupdate performed. With iTunes system, since N_1 and N_2 would like to authorize a platform, an imitate version of the platform is made available to both using perspective attributes. The implementation of a time stamp is to enable the destruction or elimination of outdated copied resources from the system.

The algorithm for a concurrent usage is formulated as follows:

Definition 1: We classify UCON as a state transition system. A Transition system is given by a tuple $T = (S, Act, \rightarrow AP, L)$, where, S is a finite or infinite set of states (state space), *Act* is a set of actions, $\rightarrow \subseteq S \times Act \times S$ is a transition relation $(s, \alpha s^1) \in \rightarrow$ is denoted by $s \xrightarrow{\alpha} s^1$, $I \subseteq S$ is a set of initial state or initial conditions, AP is a set of atomic propositions, $L : S \rightarrow 2^{AP}$ is a labeling function

Definition 2: For $s \in S$ and $\alpha \in Act$, s is the set of α, the successor S is defined as $Post(s, \alpha) = \{ s^1 \in S : s \xrightarrow{\alpha} s^1 \}$ and

the set of successors of s is defined as, $Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$

Definition 3: From definition 1, $UCON_A$ is defined as $M = (S, P_A, A_A)$, where S is a set of sequence of states, P_A is a finite set of authorization predicates and A_A is a finite set of update actions.

Definition 4: A program is defined as **racy** if two concurrent processes attempt to access the same portion of state at the same time. An example of a racy program is represented as:

x := y + x and x := z +x, where $y \neq z$

Case Study

We use iTun**es** System to illustrate concurrency and discuss our model. ITunes is a media play as well as a media library application that has been developed by Apple Inc. It is basically utilize by users to play, download and organize digital audio and video on personal computers that run the OS X operating system and iOS-based devices such as iPod, iPhone, and iPad devices. Users are able to purchase and download music, music videos, television shows, iPod games, audio books, podcasts, movies and ringtones, available on the iPhone and iPod via the iTunes Store. We identify the following objects in an iTunes system: shown in Table I. We consider a music file as a resource and represent music file with *r* in our definitions.

TABLE I
ATTRIBUTES IN UCON FOR ITUNES SYSTEM

Objects	**Attributes**	**Value**
User	Registered,	A Boolean value, A numerical value,
	Credit, orderList,	A set of resource (music) that a user has ordered,
	platformList	A platform that user authorizes to play music
iTunes server	regUsers	Users with iTunes account
Platform	authoriziedBy	A user authorizes a platform
	localList	Consist of music files stored locally in a Platform
Music file	Owner, price	The user that owns the file, Price

Definition of Terms

Let the user be represented by *U*, resource by *r*, Update Attribute by U_{AT}, Platform by *p*, Threshold by τ, Authorize platform by A_{PT}, Permit Access by ε, Null = ϕ, the binary digit 1 = True state, let the symbol ξ denotes registration, credit be *c,v* be value of the resource, Ordered Item be ω, P_L = Platform List, de-authorization platform be D_{APT}, Usage =*G*

Algorithm to Access a Resource:

Order (U, r)

i. $(U.\xi = 1) \wedge (U.c \geq r.v) \wedge (r \notin U.\omega) \rightarrow \varepsilon(U, r, \omega)$

ii. $U_{AT} : U.\omega^1 \cup \{r\}$

iii. $U_{AT} : r.U^1 = U$

iv. $U_{AT} : U.c^1 = U.c - r.v$

To utilize the resource the user has to authorize a platform. On the other hand, a user can also de-authorize the platform.

Algorithm to authorize a platform

v. $A_{PT}(U : p)$:

vi. $(U.\xi = 1) \wedge (|U.P_L| < \tau) \wedge (p \notin U.P_L) \rightarrow \varepsilon(U, p, A_{PT})$

vii. $U_{AT} : U.P_L^1 = U.P_L \cup \{p\}$

viii. $U_{AT} : p.A_{PT}^1 = U$

Algorithm for de-authorization

ix. $D_{APT} : (U, p)$:

x. $(U.\xi = 1) \wedge (p \in U.P_L) \rightarrow \varepsilon(U, p, D_{APT})$

xi. $U_{AT} : U.P_L^1 = U.P_L - \{p\}$

xii. $U_{AT} : p.A_{PT}^1 = \phi$

Finally the condition to have a resource is formulated as:

If G (p,r)
then
$(p.A_{APT} \neq \phi) \wedge (r.U \neq \phi) \wedge (p.A_{PT} = r.U) \rightarrow \varepsilon(p, r, G)$
end

VII. CONCLUSION

UCON provides a solution to the control of digital resources or objects in our computing world currently. To enhance on UCON and to accomplish its objectives in a heterogeneous environment, this paper has come out with a model by which some of the attributes and components of UCON can be created and eliminated by means of lifespan. The paper further proposed a model for the concurrent usage scenarios of UCON system. Our model gives concurrent users access to the same resource in a system without one user having to wait on the other. This is achieved by the implementation of two control agents. However due to space, we explained our model with eight core models of usage control authorization (UCONA)

REFERENCES

[1] Alnemr R, Koenig S, Eymann T, Meinel C, (2010). Enabling usage control through reputation objects: A discussion on e-commerce and the Internet of services environments. Journal of theoretical and applied electronic commerce research 5(2): 59-76.

[2] Lazouski A, Martinelli F, Mori P, (2010). Usage control in computer security: A survey. Computer Science Review, 4(2): 81-99.

[3] Basin D, Harvan M, Klaedtke F, Zalinescu E, (2011). Monitoring usage-control policiesin distributed systems. In: IEEE Eighteenth International Symposium on Temporal Representation and Reasoning (TIME), p. 88-95.

[4] Zhao B, Sandhu R, Zhang X, Qin X, (2007). Towards a times-based usage control model. In: Data and Applications Security XXI, Springer Berlin Heidelberg. p. 227-242.

[5] Maler, E, (2010). Controlling Data Usage with User-Managed Access (UMA). In: W3C Privacy and Data Usage Control Workshop, Cambridge

[6] Sastry M, Krishnan R, (2007), A new modeling paradigm for dynamic authorization in multi-domain systems. In: Computer Network Security; Springer Berlin Heidelberg, p. 153-158.

[7] Katt B, Zhang X, Breu R, Hafner M, Seifert JP, (2008). A general obligation model and continuity: enhanced policy enforcement engine for

usage control. In: Proceedings of the 13th ACM symposium on Access control models and technologies; New York, NY, USA: ACM; 2008. p. 123-132.
[8] Basin D., Harvan M., Klaedtke F and Zălinescu E, (2012). MONPOLY: Monitoring usage-control policies. In: Runtime Verification, Springer Berlin Heidelberg, 360-364.
[9] Wu J, Shimamoto S, (2010). Usage control based security access scheme for wireless sensor networks. In: 2010 IEEE International Conference on Communications (ICC), p. 1-5.
[10] Zhang X, (2006). Formal model and analysis of usage control. Ph.D. Thesis, George Mason University, Fairfax, VA, USA.
[11] Boyapati C., Lee R., and Rinard M, (2002). Ownership types for safe programming: Preventing data races and deadlocks. OOPSLA.
[12] Boyland J, (2003). Checking interference with fractional permissions. In R. Cousot, editor, Static Analysis: 10th International Symposium, volume 2694 of Lecture Notes in Computer Science, pages 55–72, Berlin, Heidelberg, New York, Springer.
[13] Dijkstra E. W, (1971) Hierarchical ordering of sequential processes. Acta Informatica, 1 2:115–138
[14] Dijkstra E. W, (1968) Cooperating sequential processes. In F. Genuys, editor, Programming Languages, pages 43–112. Academic Press.
[15] Hansen P. B; 1972; Structured multiprogramming. Comm. ACM, 15(7): 574–578
[16] Hoare C. A. R, (1972) Towards a theory of parallel programming. In Hoare and Perrot, editors, Operating Systems Techniques. Academic.
[17] O'Hearn P. W. and Pym D. J (1999) The logic of bunched implications. Bulletin of Symbolic Logic, 5(2): 215–244.
[18] Owicki S. and Gries D, (1976). Verifying properties of parallel programs: An axiomatic approach. Comm. ACM, 19(5): 279–285, 1976.
[19] Andrews G (1991); Concurrent programming: principles and practice. Benjamin/Cummings
[20] Reynolds, J. C. (2005). Toward a grainless semantics for shared-variable concurrency. In FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science (pp. 35-48). Springer Berlin Heidelberg
[21] Gotsman A.,Yang, H, (2011). Liveness-preserving atomicity abstraction. InAutomata, Languages and Programming (pp. 453-465). Springer Berlin Heidelberg.
[22] Chen, J. K., Huang, Y. F., Chin, Y. H, (1997). A study of concurrent operations on R-trees. Information Sciences, 98(1), 263-300.
[23] O'Hearn P.W, (2007). Resources, concurrency, and local reasoning. Theoretical computer science, 375(1): 271-307.
[24] Brookes S. D, (2005). A semantics for concurrent separation logic. Theoretical Computer Science, this Volume. Preliminary version appeared in Proceedings of the 15th CONCUR (2004), LNCS 3170, pp16-34.
[25] Sen K, (2008). Race directed random testing of concurrent programs. In: ACM SIGPLAN Notices 43(6): 11-21.
[26] Lu S, Tucek J, Qin F, Zhou Y, (2006). AVIO: detecting atomicity violations via access interleaving invariants. In: ACM SIGARCH Computer Architecture News, p. 37-48.
[27] Rajkumar P.V, Ghosh S.K, Dasgupta P, (2009). Application specific usage control implementation verification. International Journal of Network Security and Its Applications, 1(3):116-128.
[28] Rajkumar P.V, Ghosh S.K, Dasgupta P, (2010). Concurrent Usage Control Implementation Verification Using the SPIN Model Checke