Formal Specification of CA-UCON model using CCA

Abdulgader Almutairi
Software Technology Research Laboratory
De Montfort University
The Gateway, Leicester, LE1 9BH, UK
Abdulgader@dmu.ac.uk

François Siewe
Software Technology Research Laboratory
De Montfort University
The Gateway, Leicester, LE1 9BH, UK
fsiewe@dmu.ac.uk

***Abstract*—A Context-Aware Usage CONtrol (CA-UCON) model is an extension of the traditional UCON model which enables adaptation to environmental changes in the aim of preserving continuity of usage in a pervasive computing system. When the authorisations and obligations requirements are met by the subject and the object, and the conditions requirements fail due to changes in the environment or the system context, CA-UCON model triggers specific actions to adapt to the new situation. Besides the data protection, CA-UCON model so enhances the quality of services, striving to keep explicit interactions with the user at a minimum. This paper proposes a formal specification of the CA-UCON model in the Calculus of Context-aware Ambients (*CCA* in short). This enables formal analysis of the CA-UCON model using the execution environment of *CCA*. For illustration, some properties of the CA-UCON model are validated for a ubiquitous learning system.**

Keywords*-Pervasive system, context-aware, usage control, *CCA

I. Introduction

Usage CONtrol (UCON) model [6] is the latest major enhancement of the traditional access control models which enables *mutability* of subject and object attributes, and *continuity* of control on usage of resources. While the concept of mutability refers to the fact that attributes are not static but does change intermittently, continuity of access decision ensures that decision to permit and allow access to an object is made constantly before and during the access to an object. This access decision is based on three key factors: authorisations, obligations and conditions. Because of the continuity of access decision, access permission may be revoked as a result of changes in the environmental or system context, regardless of whether the authorisations and obligations requirements are met. This constitutes a major shortcoming of the UCON model in pervasive computing systems which constantly strives to adapt to environmental changes so as to minimise disruptions to the user.

Some recent works have been done to improve the UCON model. For instance, [11] proposed a new access control model called TUCON (Times-based Usage Control) for prevention of digital resources abuse. In TUCON a time variable is introduced into UCON, and maximum times defined as consumption constraints. This approach is easily defined in CA-UCON model by specified the time as condition requirement. [12] proposed Geography Usage Control (GEO-UCON) model to deal with GEO DBMS access control. In GEO-UCON a geospatial factor is added into UCON to ensure data security in location-based services and mobile applications. This model like the last one, can be defined in CA-UCON model where the location can be used as condition requirement to control the service. Moreover, [13] extends usage control model to context-aware in mobile computing environments. They introduced two new components into UCON model: context and states. The new model called ConUCON takes these new components plus obligations on access decisions. [14] proposed a new model called CUC model which replaces the conditions component in UCON by context and add a management module to it. The last two models are using the context only to control the resources by changing the condition component into context, which is basically the UCON model can do that.

Unlike the above UCON extensions, a Context-Aware Usage CONtrol (CA-UCON) model was proposed [1] which enables adaptation to environmental changes in the aim of preserving *continuity of usage* by triggering specific actions to adapt to new situations. In addition to data protection, CA-UCON model enhances the quality of services, striving to keep explicit interactions with the user at a minimum. This makes it more suitable for pervasive computing systems. In this paper, we present a formal specification of the CA-UCON model using the Calculus of Context-aware Ambients (*CCA* in short). Main features of *CCA* include mobility, context-awareness and concurrency. Moreover, *CCA* specifications are executable, and so enables rapid prototyping. Our main contributions are summarised as follows:

- A formal specification of CA-UCON model using *CCA* is presented (Sect. IV). This enables formal analysis of the CA-UCON model using the execution environment of *CCA*.
- An example of a ubiquitous learning system is used to demonstrate how the properties of the CA-UCON model can be validated using the execution

environment of *CCA* (Sect. V).

II. Overview of CA-UCON Model

A Context-Aware Usage CONtrol (CA-UCON) model is an extension of the traditional UCON model to enable adaptation to environmental changes in the aim of preserving continuity of access. Indeed, when the authorisations and obligations requirements are met by the subject and the object, and the conditions requirements fail due to changes in the environmental or the system context, CA-UCON model triggers specific actions to adapt to the new situation. Besides the data protection, CA-UCON model so enhances the quality of services, keeping explicit interactions with the user at a minimum. [6] defined the UCON$_{ABC}$ family core models where A stands for Authorisations, B for oBligations and C for Conditions. We defined the CA-UCON$_{ABD}$ family core models where C is replaced by D for aDaptation. So the CA-UCON$_A$ and CA-UCON$_B$ family core models are identical to UCON$_A$ and UCON$_B$, respectively. The CA-UCON$_D$ family core model comprises two models: the pre-adaptation model CA-UCON$_{preD}$ and the on-going adaptation model CA-UCON$_{onD}$.

The computational model of CA-UCON model can be described as a Finite State Machine (FSM) depicting how an subject's request to access an object is handled in the CA-UCON model. The FSM is depicted by the graph in Figure 1, where nodes are called *states* and edges are called *transitions*. The initial state, labelled *initial*, corresponds to state when the system is waiting for a suject to submit a request. There are three final states: *end*, when the access has successfully terminated; *denied*, when the access request has been denied; and *revoked*, when access permission has been revoked during access and hence the access stopped. The intuitive meaning of the remaining states of the FSM can be summarised as follows: *requesting*, denotes when the access request is being processed; *accessing*, represents the state when the actual access is taking place; *preadapting*, is the state when the system is trying to adapt to the environmental context prior to access; and finally *onadapting*, is when the system is trying to adapt to the environmental context during access.

The transitions of the FSM are labelled with the events (or actions) that fire them. The event *tryaccess* occurs when a suject sends an access request (e.g. by clicking a menu button). This event forces the FSM to enter the *requesting* state to process that access request. While in this state, the system can perform updates on subject's and object's attributes through *preupdate* events. If the authorisations, obligations and conditions requirements are all met, the system emits the *permitaccess* event and moves into the *accessing* state. If for some reasons either the authorisations requirement or the obligations requirement is not met,

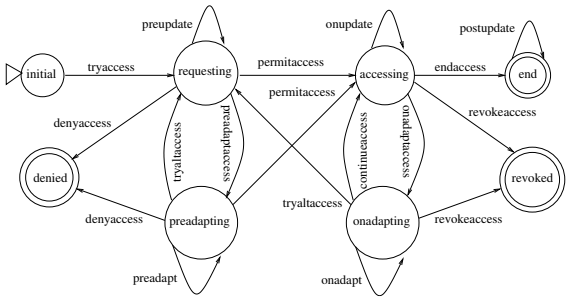

Figure 1. Execution of an access request in the CA-UCON model

the sytem emits the event *denyaccess* and terminates in the *denied* state. However, if both the authorisations requirement and obligations requirement are met, but the conditions requirement is not satisfied, the system emits the *preadaptaccess* event and moves into the *preadapting* state. In this state, specific adaptation actions, denoted by the *preadapt* events, are performed in an attempt to meet the conditions requirement. If the adaptation is successful, the *permitaccess* event is raised and the system transitions into the *accessing* state. In addition, a new request to access a specified alternative object, denoted by the *tryaltaccess* event, may be issued automatically by the system if the adaptation actions fail. Otherwise the access request is simply denied when no adaptations is possible.

When access permission is granted (see *permitaccess* event), the system transitions into the *accessing* state in which the actual access takes place. During access the system can perform updates on subject's and object's attributes via *onupdate* events. If during access either the authorisations requirement or the obligations requirement is not met, the sytem emits the event *revokeaccess* and terminates in the *revoked* state. However, if both the authorisations requirement and obligations requirement are continuously met, but the conditions requirement fails, the system raises the *onadaptaccess* event and moves into the *onadapting* state. In this state, specific adaptation actions, denoted by the *onadapt* events, are performed in an attempt to meet the conditions requirement. If the adaptation is successful, the *continueaccess* event is raised and the system moves back into the *accessing* state. In the effort to enhance the quality of service even further, the system might issue an implicit request to access a specified alternative object through the *tryaltaccess* event, when the adaptation actions fail. In the worst case when no adaptations is possible, the access permission is simply revoked and the access stopped at once.

When an access terminates successully via the *endaccess* event, the system moves into the *end* state and eventually performs updates on subject's and object's attributes through *postupdate* events. Due to the space limtation, we refer the reader to [1] for more details.

III. Overview of CCA

CCA was proposed in [5] as a process calculus for modelling mobile systems that are context-aware. It builds upon a previous calculus known as Mobile Ambients [9] whilst introducing new ideas. It enables ambient (e.g., software, devices, locations) and processes to have an awareness of the conditions and context in which they are executed. The resulting process calculus is flexible and powerful, emphasising context awareness and mobility. An ambient is an abstraction of a bounded place where computation happens. An ambient can be mobile, and can communicate with peers and can be nested inside an other ambient. In this section, we present the syntax and the informal semantics of the calculus. Due to the space limit, we refer the reader to [5] for the formal semantics of CCA. We shall define 4 syntactic categories with CCA. These are: processes P, capabilities M, locations α and context expressions κ. Names are always written in lower case letters e.g. n, x and y etc. A list of names is denoted by $\tilde{y}$ and $|\tilde{y}|$ represents the size of the list.

Table I
Syntax of CCA

P,Q	:=	$0 \mid P\|Q \mid n[P] \mid (\nu n)\ P \mid !P \mid \kappa?M.P$
M	:=	$\texttt{in}\ n \mid \texttt{out} \mid \alpha\ x\langle\tilde{y}\rangle \mid \alpha\ (\tilde{y}) \mid \alpha\ \langle\tilde{y}\rangle$
α	:=	$\uparrow \mid n\uparrow \mid \downarrow \mid n\downarrow \mid :: \mid n :: \mid \epsilon$
κ	:=	$\texttt{true} \mid \bullet \mid n = m \mid \neg\kappa \mid \kappa_1 \mid \kappa_2$
		$\mid \kappa_1 \wedge \kappa_2 \mid new\ n, \kappa \mid \oplus\kappa \mid \Diamond\kappa$

Processes: The process 0 terminates immediately. If two processes P and Q are running in parallel, this is denoted by $P|Q$. To limit the scope of a name, the following notation is used: $(\nu n)\ P$, indicating that the scope of n is limited to P. The replication $!P$ denotes a process that can recreate a copy of itself whenever needed, i.e. $!P \equiv P|!P$. The process $n[P]$ represents an ambient named n whose behaviour is described by the process P.

A context expression κ denotes the situation that must be met by the environment of the executing process. The context-guarded prefix $k?M.P$ is a process that waits until the environment satisfies the context-expression κ, then executes the capability M and continues like the process P.

Locations: The location α can be '$\uparrow$ ' which indicates any parent, '$n\uparrow$' for a definite parent ambient named n, '$\downarrow$' which denotes any child, and '$n\downarrow$' for a definite child ambient named n, the '$::$' which refers to any sibling, and '$n::$' signifies a specific sibling ambient named n. The symbol ϵ (empty string) refers to the current ambient.

Capabilities: There are two mobility capabilities defined in CCA [5], which make it possible for an ambient to move in its environment. These are '`in`' and '`out`'. An ambient can execute the capability '`in` n' to move into a sibling ambient named n, and the capability `out` allows an ambient to move out of its parent ambient. Ambients are able to send and receive messages. Using the capability $\alpha\ \langle\tilde{y}\rangle$ an ambient is able to send a list of names $\tilde{y}$ to a location α. An ambient can execute the capability $\alpha\ (\tilde{y})$ to receive in the variables in $\tilde{y}$ a set of names from a location α.

Context expression: In *CCA*, a context is modelled as a process with a hole in it. The hole (denoted by $\odot$) in a context represents the position of the process that context is the context of. For example, suppose a system is modelled by the process $P \mid n[Q \mid m[R \mid S]]$. So, the context of the process R in that system is $P \mid n[Q \mid m[\odot \mid S]]$, and that of the ambient named m is $P \mid n[Q \mid \odot]$. Properties of contexts are called context expressions (CE in short).

The CE `true` holds for all context. A CE $n = m$ holds if the names n and m are lexically identical. The CE $\bullet$ holds solely for the hole context, i.e. the position of the process evaluating that context expression. Propositional operators such as negation ($\neg$) and conjunction ($\wedge$) expand their usual semantics to context expressions. A CE $\kappa_1|\kappa_2$ holds for a context if that context is a parallel composition of two contexts. A CE $n[\kappa]$ holds for a context if that context is an ambient named n such that κ holds inside that ambient. A CE $\oplus\kappa$ holds for a context on the condition that the context has a child context for which κ holds. For a CE $\Diamond\kappa$ to hold for a context there must be a sub-context present, somewhere in that context, for which κ holds. The operator $\Diamond$ is known as *somewhere modality* while $\oplus$ is known as *spatial next modality*.

IV. Formalising the CA-UCON Model in CCA

In this section, we give a formalisation of the CA-UCON model using mathematical notation of *CCA*. We model each non-terminal state of the automaton in Figure 1 as an ambient. The behaviour of the user that initiates an access request is also represented by an ambient named *subject*. In addition, specific ambients are used to check the authorisation, obligation and condition requirements. So the formal specification of the CA-UCON model is given by the following *CCA* process:

$$\begin{aligned} &subject[P_{subject}] \mid requesting[P_{requesting}] \\ &\mid accessing[P_{accssing}] \mid preadapting[P_{preD}] \\ &\mid onadapting[P_{onD}] \mid checkPreA[P_{preA}] \\ &\mid checkPreB[P_{preB}] \mid chechPreC[P_{preC}] \\ &\mid checkOnA[P_{onA}] \mid checkOnB[P_{onB}] \\ &\mid checkOnC[P_{onC}] \end{aligned}$$

The specifications of these ambients are detailed in the following subsections. First, we give in Table II a convention of notations used in these specifications; these include constants and variables symbols.

Table II
CONSTANTS AND VARIABLES

Constants	
Notation	Description
PERMIT	permit access
DENYA	deny Authorisation
DENYB	deny Obligation
DENYC	deny Condition
REVOKEA	revoke Authorisation
REVOKEB	revoke Obligation
REVOKEC	revoke Condition
END_USAGE	terminate usage
ENDED_SUCCESSFULLY	usage ended successfully

Variables		
Notation	Description	values
s	user id	201,202, 203
o	object id	lect1, tut3, test4
r	access right	download, read, write
preA	Preauthorization	0,1
preB	Preobligation	0,1
preC	Precondition	0,1
preD	Preadaptation	0,1
onA	Onauthorisation	0,1
onB	Onobligation	0,1
onC	Oncondition	0,1
onD	Onadaptation	0,1

A. Subject Ambient

This ambient is responsible for submitting the access request to the *requesting* ambient; it then waits for the reply to this access request, whether *permit* or *deny*. The user eventually ends the usage by sending the message END_USAGE to the *accessing* ambient. This behaviour is modelled as follows:

$$P_{subject} \widehat{=} !requesting :: \langle s,o,r\rangle.requesting :: (reply,o,r).\{ (reply = PERMIT)?accessing :: (x,o,r).0 \mid accessing :: \langle \text{END\_USAGE}, o, r\rangle.0 \}$$

where s is the user id, o the object id and r the access right requested.

B. Requesting Ambient

This ambient handles all the access requests sent by the subject in order to access an object. It receives an access request from the subject ambient and checks the *preauthorisation*, *preobligation* and *precondition* by sending the access request to the *checkPreA*, *checkPreB* and *checkPreC* ambients, respectively. If the *preA* and *preB* and *preC* are true (i.e. 1), the subject is permitted to access the requested object. But if *preA* (or *preB*) is not true (i.e. 0), the reply *DENYA* (or *DENYB* respectively) is sent to the subject. However, if *preA* and *preB* are true but *preC* is false, the request is sent to the *checkPreD* ambient in order to adapt to the new situation if possible. The reply *DENYC* is sent to the subject if the adaptation fails (i.e. $preD = 0$). This behaviour is modelled as follows:

$$\begin{array}{l} P_{requesting} \widehat{=}\ !::(s,o,r).checkPreA :: \langle s,o,r\rangle. \\ checkPreA :: (preA). \\ \left\{ \begin{array}{l} (preA=1)?checkPreB :: \langle s,o,r\rangle. \\ checkPreB :: (preB). \\ \left\{ \begin{array}{l} (preB=1)?checkPreC :: \langle s,o,r\rangle. \\ checkPreC :: (preC). \\ \left\{ \begin{array}{l} (preC=1)?subject :: \langle PERMIT,o,r\rangle. \\ preUpdate :: \langle s,o,r\rangle.preUpdate :: (). \\ accessing :: \langle s,o,r\rangle.0 \\ \mid (preC=0)?preadapting :: \langle s,o,r\rangle. \\ preadapting :: (preD).\{ \\ \quad (preD=1)?subject :: \langle PERMIT,o,r\rangle. \\ \quad preUpdate :: \langle s,o,r\rangle. \\ \quad preUpdate :: ().accessing :: \langle s,o,r\rangle.0 \\ \quad \mid (preD=0)?subject :: \langle DENYC,o,r\rangle.0 \\ \} \end{array} \right\} \\ \mid (preB=0)?subject :: \langle DENYB,o,r\rangle.0 \end{array} \right\} \\ \mid (preA=0)?subject :: \langle DENYA,o,r\rangle.0 \end{array} \right\} \end{array}$$

C. Accessing Ambient

This ambient is responsible for continuously controlling all the permitted access requests that it receives from the *requesting* ambient. This is achieved through sending the access request to the *checkOnA*, *checkOnB* and *checkOnC* ambient, and receiving the reply in the following variables: *onA*, *onB* and *onC*. If *onA* or *onB* is false, the reply *REVOKEA* or *REVOKEB* is sent to the subject, respectively. However, if *onA* and *onB* are true but *onC* is false, the access request is sent to *checkOnD*, whereupon it receives the parameter *onD*; if this is true, the subject will be permitted to continue to access the object until the user ends the usage. Otherwise, the reply *REVOKEC* is sent to the subject ambient and the usage terminates immediately. This behaviour is modelled as follows:

$$\begin{array}{l} P_{accessing} \widehat{=}\ !::(s,o,r). \\ checkOnA :: \langle s,o,r\rangle.checkOnA :: (onA). \\ \left\{ \begin{array}{l} (onA=1)?checkOnB :: \langle s,o,r\rangle.checkOnB :: (onB). \\ \left\{ \begin{array}{l} (onB=1)?checkOnC :: \langle s,o,r\rangle. \\ checkOnC :: (onC). \\ \left\{ \begin{array}{l} (onC=1)?onUpdate :: \langle s,o,r\rangle. \\ onUpdate :: ().accessing :: \langle s,o,r\rangle.0 \\ \mid (onC=0)?onadapting :: \langle s,o,r\rangle. \\ onadapting :: (onD).\{ \\ \quad (onD=1)?onUpdate :: \langle s,o,r\rangle. \\ \quad\quad onUpdate :: ().accessing :: \langle s,o,r\rangle.0 \\ \quad \mid (onD=0)?subject :: \langle REVOKEC,o,r\rangle.0 \\ \} \end{array} \right\} \\ \mid (onB=0)?subject :: \langle REVOKEB,o,r\rangle.0 \end{array} \right\} \\ \mid (onA=0)?subject :: \langle REVOKEA,o,r\rangle.0 \end{array} \right\} \\ \mid subject :: (x).postUpdate :: ().postUpdate :: \langle s,o,r\rangle. \\ postUpdate :: (x).subject :: \langle ENDED\_SUCCESSFULLY,o,r\rangle.0 \end{array}$$

D. Preadapting Ambient

When the *precondition* does not hold, the *requesting* ambient forwards the access request to this ambient which then attempts to adapt to the context by performing specific actions. In addition, it is able to issue an alternative request which is appropriate to the current situation. Otherwise, the access request is simply *denied*, when no adaptation is possible. This behaviour is modelled as follows:

$$P_{preD} \widehat{=}\ !requesting :: (s,o,r).P.requesting :: \langle preD\rangle.0$$

where *P* stands for the pre-adaptation actions or alternative request, which must be done by the *preadapting* ambient. The decision is calculated in the variable *preD* and sent to the *requesting* ambient. The actual specification of the process P is application dependent.

E. Onadapting Ambient

When the *oncondition* is false during the access, the *accessing* ambient forwards the access request to the *onadapting* ambient which attempts to adapt to the new situation in order to maintain continuity of usage of the resource. Like the *requesting* ambient, it is capable of issueing an alternative request depending on the context. If no adaptation is possible, the access is simply revoked. This behaviour is modelled as follows:

$$P_{onD} \widehat{=}\ accessing :: (s, o, r).P.accessing :: \langle onD \rangle.0$$

where *P* stands for on-adaptation actions or alternative request, which must be performed by the *onadapting* ambient. The decision is calculated in the variable *onD* and sent to the *accessing* ambient.

F. CheckPreA Ambient

This ambient receives an access request from the *requesting* ambient and checks whether the pre-authorisation requirements are met by the subject. It then sends the decision *preA* to the *requesting* ambient. This behaviour is achieved as follows:

$$P_{preA} \widehat{=}\ !requesting :: (s, o, r).P.requesting :: \langle preA \rangle.0$$

where the process *P* models the pre-authorisation requirements, which again are application dependent.

G. CheckPreB Ambient

This ambient receives an access request from the *requesting* ambient and checks whether the preobligation requirements are met by the subject. Then, it sends the decision *preB* to the *requesting* ambient. This behaviour is achieved as follows:

$$P_{preB} \widehat{=}\ !requesting :: (s, o, r).P.requesting :: \langle preB \rangle.0$$

where the process *P* models the preobligation requirements.

H. CheckPreC Ambient

This ambient receives an access request from the *requesting* ambient and checks whether the *precondition* requirements are met by the environment. Then, it sends the decision *preC* to the *requesting* ambient. This behaviour is achieved as follows:

$$P_{preC} \widehat{=}\ !requesting :: (s, o, r).P.requesting :: \langle preC \rangle.0$$

where the process *P* represents the precondition requirements.

I. CheckOnA Ambient

This ambient plays similar role for *onauthorisation* requirements like the *checkPreA* ambient for *preauthorisation* requirements. The decision whether the *onauthorisation* requirements are met is returned to the *accessing* ambient via the variable *onA*. This behaviour is specified as follows:

$$P_{onA} \widehat{=}\ !accessing :: (s, o, r).P.accessing :: \langle onA \rangle.0$$

where the process *P* models the onauthorisation requirements.

J. CheckOnB Ambient

Similarly to the *checkOnA* ambient, this ambient plays the same role for *onobligation* requirements as the *checkPreB* ambient for *preobligation* requirements. The decision whether the *onobligation* requirements are met is returned to the *accessing* ambient via the variable *onB*. This behaviour is specified as follows:

$$P_{onB} \widehat{=}\ !accessing :: (s, o, r).P.accessing :: \langle onB \rangle.0$$

where *P* represents the *onobligation* requirements.

K. CheckOnC Ambient

As for this ambient, it receives an access request from the *accessing* ambient and checks whether the on-condition requirements are met by the environment. The decision is sent to the *accessing* ambient in the variable *onC*. This behaviour is specified as follows:

$$P_{onC} \widehat{=}\ !accessing :: (s, o, r).P.accessing :: \langle onC \rangle.0$$

where the process *P* models the on-condition requirements.

V. Validation

In this section, we illustrate how CA-UCON model properties can be validated using the execution environment of *CCA*. This is done via an example of a ubiquitous learning system (u-learning).

A. Example: a u-learning system

A u-learning system takes into account the context of the learners, their devices, and the environment in order to provide them services such as u-lectures, u-tutorials and u-tests. It enables learners to use their portable devices, such as smart phones, laptops and PDAs, to connect wirelessly to various wireless networks, such as Wi-Fi spots, WLANs and 3G terminals, in order to access u-learning services anytime, anywhere. The u-learning system delivers the content of services to the learner as appropriate and adaptable content, based on the context of use (e.g. user devices, available bandwidth, and user activities). These services are available in three formats: text, audio and video. The policies of u-learning system are presented as follows:

- The u-learning system considers the context along with the type of requested u-service before and during the access request. Therefore, the u-learning system would allow the learner to access only one type of u-service format if the context of the learner were to be detected as 'driving', which would be the audio format. However, if the context of the

learner were 'not driving', s/he would be allowed to access any u-service format.

- The location context of the learner is classified as private or public. The public context is defined as learners being in a location where making noise is not permitted, such as in a library, seminar or lecture hall. On the other hand, the private context denotes that the learners are in a location where making noise is permitted, such as in a cafe or at home. The only u-service format allowed for a learner in a public context is the text format. However, the system permits access to all the different u-service formats when the learner context is detected as private.
- The available memory on the user's device is also important. Each u-service format requires a different amount of space for memory space in order to facilitate access. If the learner requests a u-service in the video format, the available memory of the device must be more than 5MB for access to be permitted. However, the available memory of the device must only be more than 2MB if an audio is requested, and more than 1MB for a text.

B. Executing Scenarios

In following paragraphs, we design two scenarios and execute them in order to validate some properties of CA-UCON model in u-learning system. The two main properties are safety property and liveness property:

Scenario 1: The property to validate in this scenario is : *" if the pre-authorization, pre-obligation and pro-condition requirements are met at the time of access request and on-authorization, on-obligation and on-condition requirements are continuously hold, the access will be successful"*. Suppose a learner requests to download a u-lecture in the video format and the context of learner is in *private* place and the memory capacity of her/his mobile device is more than 5MB. The service will be delivered to the learner by the system.

Figure 2 presents a screen shot of the execution of Scenario 1, from which it can be seen that the subject sends the access request to the *requesting* ambient (line 2). The *requesting* ambient receives the access request and checks the *pre-authorisation, pre-obligation* and *pre-conditions* pertaining to this request (lines 3-8). The ambient *checkPreC* checks the context of the subject, the memory capacity of the mobile device and the bandwidth of the network before access. In this case, all the requirements before the access are met and the access is *permit* to the service(lines 16). Moreover, the *requesting* ambient sends the access request to *accessing* ambient to check *on-authorisation, on-obligation* and *on-conditions* pertaining to this request, and all requirements are hold during the access (lines 22-27). In

```
CCA Parser Version 4.01:  Reading from file ca_ucon6.cca . . .
CCA Parser Version 4.01:  CCA program parsed successfully.
Execution mode: interleaving
1 --->  {Sibling to sibling: subject ===(P1,video,download)===> requesting}
2 --->  {Sibling to sibling: requesting ===(P1,video,download)===> checkpreA}
3 --->  {Sibling to sibling: checkpreA ===(1)===> requesting}
4 --->  {Sibling to sibling: requesting ===(P1,video,download)===> checkpreB}
5 --->  {Sibling to sibling: checkpreB ===(P1)===> UP}
6 --->  {Sibling to sibling: UP ===(1)===> checkpreB}
7 --->  {Sibling to sibling: checkpreB ===(1)===> requesting}
8 --->  {Sibling to sibling: requesting ===(P1,video,download)===> checkpreC}
9 --->  {Child to parent: checkpreC ===(P1,context)===> SubjectCxt}
10--->  {Child to parent: SubjectCxt ===(P1,private)===> checkpreC}
11--->  {Child to parent: checkpreC ===(P1,fm)===> MemorySize}
12--->  {Child to parent: MemorySize ===(P1,6)===> checkpreC}
13--->  {Child to parent: checkpreC ===(P1,bandwidth)===> Bandwidth}
14--->  {Child to parent: Bandwidth ===(P1,high)===> checkpreC}
15--->  {Sibling to sibling: checkpreC ===(1)===> requesting}
16--->  {Sibling to sibling: requesting ===(PERMIT,video,download)===> subject}
17--->  {Sibling to sibling: requesting ===(P1,video,download)===> preUpdate}
18--->  {Sibling to sibling: preUpdate ===(5)===> francois_credit}
19--->  {Sibling to sibling: francois_credit ===()===> preUpdate}
20--->  {Sibling to sibling: preUpdate ===(done)===> requesting}
21--->  {Sibling to sibling: requesting ===(P1,video,download)===> accessing}
22--->  {Sibling to sibling: accessing ===(P1,video,download)===> checkonA}
23--->  {Sibling to sibling: checkonA ===(1)===> accessing}
24--->  {Sibling to sibling: accessing ===(P1,video,download)===> checkonB}
25--->  {Sibling to sibling: checkonB ===(1)===> accessing}
26--->  {Sibling to sibling: accessing ===(P1,video,download)===> checkonC}
27--->  {Sibling to sibling: checkonC ===(1)===> accessing}
28--->  {Sibling to sibling: accessing ===(P1,video,download)===> onUpdate}
29--->  {Sibling to sibling: onUpdate ===(1)===> francois_credit}
30--->  {Local: francois_credit ===(5)===> francois_credit}
31--->  {Sibling to sibling: francois_credit ===()===> onUpdate}
32--->  {Sibling to sibling: onUpdate ===(done)===> accessing}
33--->  {Sibling to sibling: subject ===(END_USAGE,video,download)===> accessing}
34--->  {Sibling to sibling: accessing ===(P1,video,download)===> postUpdate}
35--->  {Sibling to sibling: postUpdate ===(0)===> francois_credit}
36--->  {Sibling to sibling: francois_credit ===()===> postUpdate}
37--->  {Sibling to sibling: postUpdate ===(done)===> accessing}
38--->  {Sibling to sibling: accessing ===(ENDED_SUCCESSFULLY,video,download)===> subject}
Done
```

Figure 2. Execution of Scenario1

line(33) The subject ended the access successfully by sending this message END_USAGE to the *accessing* ambient.

Scenario 2: the property to validate in this scenario is : *" if the one of the pre-authorization or pre-obligation requirements are not met before the access is permit, the access will be denied by the system"*. Suppose a learner requests to download a u-lecture in the video format and the context of learner is in *private* place and the memory capacity of her/his mobile device is more than 5MB. However, the pre-authorization requirements of this access request are not met. In this case, the system has to deny the access request.

```
CCA Interpreter version 4.01
October 2012
Please send error messages to
- fsiewe@dmu.ac.uk
- fsiewe@yahoo.fr
CCA Parser Version 4.01:  Reading from file ca_ucon6.cca . . .
CCA Parser Version 4.01:  CCA program parsed successfully.
Execution mode: interleaving
1--->  {local call to the abstraction "mem" in the ambient "root"}
2--->  {Sibling to sibling: subject ===(P1,video,download)===> requesting}
3--->  {Sibling to sibling: requesting ===(P1,video,download)===> checkpreA}
4--->  {Sibling to sibling: checkpreA ===(0)===> requesting}
5--->  {Sibling to sibling: requesting ===(denyA,video,download)===> subject}
```

Figure 3. Execution of Scenario 2

Figure 3 illustrates the execution of scenario 2. The subject sends the access request containing the subject ID, the u-lecture in the video format and the downloading right to the *requesting* ambient (line 2). The *requesting* ambient receives the access request and checks the *pre-authorisation* pertaining to this request (line 3). So, the *pre-authorization* of this request is not

met, and finally the system denied the access request (lines 4-5).

VI. Related Work

There are many existing formal specification which support mobility and/or context-aware, but majority of them were not appropriated in order to model context-aware mobile applications. For instance, one of these formal specification is called CONAWA proposed by [2] as calculus for applications that are context-aware, and this formal specification is inspired by π calculus. The CONAWA's syntax concentrates on constructs that make it probable to navigate and describe via context. Another formal specification was proposed by [3] which is known as Bigraphs, it is a unifying framework that used to model concurrent mobile systems, but it does not support context-awareness. Moreover, a UML specification of the infostation-based mLearning system was proposed by [4]. Despite the fact that UML specification offers a number of benefits like system analyses utilizing assistant tools and code generation, it has suffered from the need of formal reasoning support which represent its limitation for the design of critical systems. As consequence, we select *CCA* in order to model a context-aware system, as it is a mathematical notation that support mobility and context-awareness, as well as treats those primitive constructs as first-class citizens. However, to the best of our knowledge, none of these formal specification or others are used to model CA-UCON model.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented the formalisation of a context-aware usage control in *CCA*. A CA-UCON model is represented as a process and so can be executed and analysed using the execution environment of *CCA*. We have illustrated how such analysis can be done using an example of a u-learning system. Scenarios can be designed and executed to test various properties of the model.

In future works, we will investigate possible enforcement mechanisms of the CA-UCON model in a pervasive environment.

References

[1] A. Almutairi and F. Siewe. *CA-UCON: A Context-Aware Usage Control Model*.In Proceeding of the 5th ACM International Workshop on Context-Awareness for Self-Managing Systems[CASEMANS '11],pp.34-38,2011.

[2] M. B. Kj ergaard and J. Bunde-Pedersen. CONAWA: *A Formal Model for Context Awareness*. Technical Report 0909-0878, Basic Research in Computer Science BRICS, 2006.

[3] R. Milner. *Pure Bigraphs: Structure and Dynamics*. Information and Computation/information and Control, 204:60-122, 2006.

[4] I. Ganchev, S. Stojanov, and D. Meere. *An InfoStation-Based Multi-Agent System Supporting Intelligent Mobile Services Across a University Campus*. Journal of Computers, 2(3), 2007.

[5] F. Siewe, H. Zedan, and A. Cau. *The Calculus of Context-aware Ambients*.Journal of Computer and System Sciences,77(4):597-620, 2011.

[6] P. Jaehong and S. Ravi. *The UCON: Usage Control Model*. Journal of ACM Transactions on Information and System Security,V.7,PP 128-174,No.1, 2004.

[7] I. Horvath and D. Peck. *Survey of Advanced Learning Solutions from Methodological and technological perspectives*. In proceedings of 24th ASCILITE Conference, Singapore,2009.

[8] S. Almutairi, H. Aldabbas and A. Abu-Samaha. *Review on the security related issues in context aware system*. International Journal of Wireless & Mobile Networks(IJWMN),Vol.4, No,3,PP.195-204, 2012.

[9] L. Cardelli and A. Gordon. *Mobile ambients*. Theoretical Computer science, 240: 177-213,2000.

[10] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[11] B. Zhao,S. Ravi, X. Zhang and X. Qin *Towards a Times-Based Usage Control Model*. in international Federation for Information Processing,2007. S. Barker, G.-J.Ahn(Eds):Data and Applications Security 2007, LNCS 4602,p.227-242.

[12] Z. Hong-jun. *Study and application of special access control model based on UCON*. Master Thesis. Nanjing: Jiangshu University 2009.

[13] G.Bai, L.Gu, T. Feng, Y.Guo and X. Chen. *Context-Aware Usage Control for Android*. S. Jojodia and J. Zhou (Eds): SecureComm 2010, LNICST 50,p.326-343.[DOI:10.11007/978-3-642-16161-2].

[14] L. Xiaoofeng, L. Ling, M. Ail and L. Wanbo. *The Contextual Usage Control Model*. Journal of Zhejiang University Science C (Computers & Electronics) ZUSC-D-12-00217. 2012.