Formal Specification of an Intelligent Message Notification Service in an Infostation-based mLearning System using CCA

Mohammed Al-Sammarraie, François Siewe, Hussein Zedan
Software Technology Research Laboratory
De Montfort University
Leicester - United Kingdom
{mhd, fsiewe, hzedan}@dmu.ac.uk

Abstract

The Calculus of Context-aware Ambients (CCA in short) has been proposed as a notation that is suitable to model mobile applications that are context-aware. This paper considers a real-world case study of an infostation-based mLearning system in which mobile devices such as hand-set phones, PDA's and laptops can access a number of services and communicate to each other within a university campus. Such a dynamic system must enforce complex policies to cope with mobility and context-awareness. We show how policies can be formalised using CCA, and validated using the execution environment of CCA. We illustrate how properties can be validated using our approach.

Keywords: Context-awareness, mLearning, infostation, ambient, IMN

1 Introduction

The world is increasingly becoming a larger network fully enhanced with different sensors and smart agents that have the ability to detect, sense and adapt to changes in people, devices, locations, applications and events. The topology of such a network changes dynamically in an unpredictable manner. This poses many challenges faced by the context-aware computing community. However, for the modelling of such systems, the Calculus of Context-aware Ambients (CCA in short) has been proposed as a suitable mathematical notation for modelling mobile context-aware systems. In CCA, mobility and context-awareness are primitive constructs.

An infostation-based mLearning system is considered throughout the work of this paper. This system enables mobile devices to access a number of services and communicate between each other across a university campus. Such a dynamic system must enforce complex policies to cope with mobility and context-awareness. In terms of QoS, it is essential for such a system to be modelled and its properties to be validated using a sound mathematical notation. In this paper, we are dealing with the Intelligent Message Notification (IMN) service, in which we will show how to formally specify its policies in a natural fashion using CCA. Our contributions are:

- Elicitation of the policies of the IMN service (Section 2);
- Formalisation of these policies (Section 4); and
- Validation of a liveness property of the mLearning system using the execution environment of CCA for illustration (Section 5).

2 Infostation-based mLearning system

eLearning (including mobile learning) is becoming a possible alternative approach to the traditional face-to-face learning. The infostation paradigm first proposed by Frenkiel *et al.* [3] and used in [4] to devise an infostation-based mLearning system which allows mobile devices such as hand-set phones, laptops and personal digital assistants (PDAs) to communicate to each other and to a number of services within a university campus. This section presents the architecture and the policies of the IMN service.

2.1 mServices

The mobile service (mServices) provided by the infostation-based mLearning system are:

- Authentication, Authorisation and Accounting (AAA);
- mLecture;

- mTest;
- mTutorial;
- Intelligent Message Notification (IMN); and
- VoIP

In this paper, we will focus on the IMN service only.

2.2 Architecture

The architecture of the infostation-based mLearning systems is depicted in Figure 1.

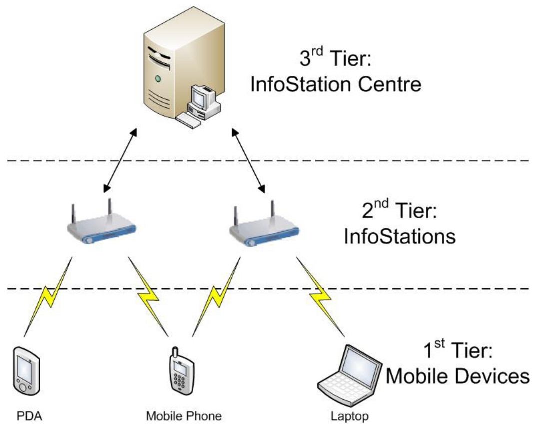


Figure 1. The three tiers architecture of the InfoStation-based network [4]

The first tier encompasses mobile devices. Each mobile device runs an intelligent agent that operates as a personal assistant (PA) for the users. The second tier consists of Infostations (ISs in short), deployed irregularly (i.e. geographically) around a university campus. These ISs enhance the operation of supplying the users access to the mServices [4]. ISs are equipped to support different communication protocols to provide a successful access to a wide range of mobile devices. The third tier is the InfoStation Centre (ISC in short). Its main function is controlling the ISs, updating and synchronising the information across the system. It also manages the users' and services' profiles.

2.3 IMN Service Policies

Via the IMN service, users can exchange text messages between each other. A user can send a message to another user located in the same or different IS of that user.

When a user walks into an IS, he first has to register with the system in order to use any of the available services. A user may choose to send a message to another user, the following policies apply:

- The sender device forwards an IMN request to the IS, which forwards the request to the ISC. The ISC checks to see whether the sender or the recipient are eligible for using this service or not (i.e. not using the mTest service).
- A user cannot use the IMN service and the mTest Service simultaneously. The mTest service should operate unaccompanied at all occasions.
- If the sender or the recipient is using the mTest service, then the request to communicate via IMN service will be denied by sending a DENIED message to the sender.

Figure 2 below depicts the architecture of the infostation-based network.

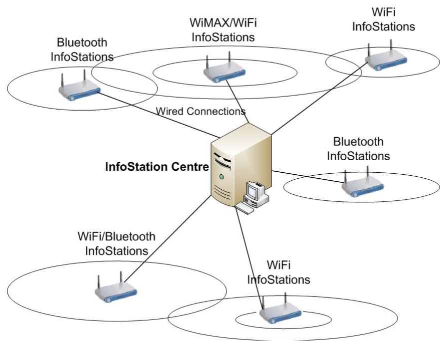


Figure 2. The architecture of the infostation-based system [4]

3 CCA: The Calculus of Context-aware Ambients

CCA [6] was proposed as a process calculus for modelling mobile systems that are context-aware. It is based on the notion of *ambient* as defined in the calculus of Mobile Ambients (MA for short) [2] and provides primitives for modelling mobility and context-awareness. We have chosen this formal method to specify our work because of CCA's modularity and the reasons that it can model mobile, context-aware, concurrent systems.

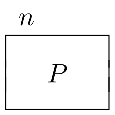
An ambient is an abstraction of a bounded place where computation happens. An ambient can be mobile, and can communicate with peers and can be nested inside an other ambient. In this section, we present the syntax and the informal semantics of the calculus. Due to the space limit, we refer the reader to [6] for the formal semantics of CCA, and how CCA is different from other similar process calculi.

Like in the π-calculus [5, 7], the simplest entities of the calculus are *names*. These are used to name for example ambients, locations, and devices. We assume a countably-infinite set of names, elements of which are written in lower-case letters, e.g. n, x and y. We let $\tilde{y}$ denote a list of names and $|\tilde{y}|$ the arity of such a list. We sometimes use $\tilde{y}$ as a set of names where it is appropriate. As depicted in Table 1, we distinguish three main syntactic categories: processes P, capabilities M and context expressions κ (read *kappa*).

Table 1. Syntax of CCA

$$\begin{array}{rcl}
P, Q & ::= & \mathbf{0} \mid P|Q \mid n[P] \mid !P \mid \kappa?M.P \mid x \triangleright (\tilde{y}).P \\
M & ::= & \mathtt{in}\, n \mid \mathtt{out} \mid \alpha\, x\langle\tilde{y}\rangle \mid \alpha\ (\tilde{y}) \mid \alpha\ \langle\tilde{y}\rangle \mid \mathtt{del}\, n \\
\alpha & ::= & \uparrow \mid n\uparrow \mid \downarrow \mid n\downarrow \mid\ ::\ \mid n\ ::\ \mid \epsilon \\
\kappa & ::= & \mathbf{True} \mid \bullet \mid n = m \mid \neg\kappa \mid \kappa_1|\kappa_2 \mid \kappa_1 \wedge \kappa_2 \mid \\
 & & \oplus\kappa \mid ✧\kappa \mid \exists x.\kappa
\end{array}$$

Processes The process $\mathbf{0}$ does nothing and terminates immediately. The process $P|Q$ denotes the process P and the process Q running in parallel. The process $(\nu n)\ P$ states that the scope of the name n is limited to the process P. The replication $!P$ denotes a process which can always create a new copy of P, i.e. $!P \equiv P|!P$. The process $n[P]$ denotes an ambient named n whose behaviours are described by the process P. The pair of square brackets '[' and ']'outlines the boundary of that ambient. This is the *textual* representation of an ambient. The *graphical* representation of that ambient is:



The graphical representation highlights the nested structure of ambients.

A context expression specifies the condition that must be met by the environment of the executing process. A *context-guarded prefix* $\kappa?M.P$ is a process that waits until the environment satisfies the context expression κ, then performs the capability M and continues like the process P. The use of context-guarded prefix is one of the two main mechanisms for context acquisition in CCA (the second mechanism for context acquisition is the call to a process abstraction as discussed below). We let $M.P$ denote the process $\mathbf{True}?M.P$, where $\mathbf{True}$ is a context expression satisfied by all context. A process abstraction $x \triangleright (\tilde{y}).P$ denotes the linking of the name x to the process P where $\tilde{y}$ is a list of *formal parameters*. This linking is local to the ambient where the process abstraction is defined. So a name x can be linked to a process P in one ambient and to a different process Q in another ambient. A call to a process abstraction named x is done by a capability of the form $\alpha\ x\langle\tilde{z}\rangle$ where α specifies the location where the process abstraction is defined and $\tilde{z}$ is the list of *actual parameters*. The location α can be '$\uparrow$' for any parent, '$n\uparrow$' for a specific parent n, '$\downarrow$' for any child, '$n\downarrow$' for a specific child n, '::' for any sibling, 'n ::' for a specific sibling n, or ϵ (empty string) for the calling ambient itself. In the hierarchy of ambients, a parent ambient denotes an ambient which is one level higher than the current one, in the same context, a child ambient indicates an ambient which is one level lower than the current ambient, while a sibling ambient denotes an ambient which lies at the same level of the current ambient. A process call $\alpha\ x\langle\tilde{z}\rangle$ behaves like the process linked to x at location α, in which each actual parameter in $\tilde{z}$ is substituted for each occurrence of the corresponding formal parameter. A process call can only take place if the corresponding process abstraction is *available* at the specified location.

Capabilities Ambients exchange messages using the capability $\alpha\ \langle\tilde{z}\rangle$ to send a list of names $\tilde{z}$ to a location α, and the capability $\alpha\ (\tilde{y})$ to receive a list of names from a location α. The mobility capabilities `in` and `out` are defined as in MA [2]. An ambient that performs the capability $\mathtt{in}\ n$ moves into the sibling ambient n. The capability `out` moves the ambient that performs it out of that ambient's parent. The capability $\mathtt{del}\ n$ deletes an ambient of the form $n[\mathbf{0}]$ situated at the same level as that capability, i.e. the process $\mathtt{del}\ n.P \mid n[\mathbf{0}]$ reduces to P. The capability `del` acts as a garbage collector that deletes ambients which have completed their computations.

Example 3.1 *A persistent cell is a data structure that operates through two actions put and get, respectively putting one item in the cell and taking a copy of the item in it. The cell keeps its content after a get action is performed. However, a put action replaces the item in the cell with a new one. A persistent cell can be modelled by the following ambient where 5 is the initial value in the cell:*

$$cell \left[\begin{array}{ll} & \langle 5\rangle.\mathbf{0} \\ \mid & !\uparrow().(w).(\langle w\rangle \mid \uparrow\langle w\rangle).\mathbf{0} \\ \mid & !\uparrow(x).(y).(\langle x\rangle \mid \uparrow\langle\rangle).\mathbf{0} \end{array} \right] \qquad (1)$$

A get action is performed as followed, where y is a variable that will receive the value stored in the cell and may occur free in the continuation process P:

$$cell\downarrow\langle\rangle.\mathbf{0} \mid cell\downarrow(y).P \mid Eq.\ (1).$$

Similarly, a put action is performed as follows, where 7 is the value to put in the cell and Q is a continuation process:

$$cell\downarrow\langle 7\rangle.\mathbf{0} \mid cell\downarrow().Q \mid Eq.\ (1).$$

Context model 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 ambient named m is $P \mid n[Q \mid \odot]$. Thus the contexts of CCA processes are described by the grammar in Table 2. Properties of contexts are called context expressions (CEs in short).

Table 2. Syntax of contexts

$$E \quad ::= \quad \mathbf{0} \mid \odot \mid n[E] \mid E|P \mid (\nu n)\ E$$

Context expressions The CE **True** always holds. 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. First order operators such as negation ($\neg$), conjunction ($\wedge$) and existential quantification ($\exists$) 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 such that κ_1 holds for one and κ_2 holds for the other. 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 $\texttt{new}(n, \kappa)$ holds for a context if that context is a restriction of the name n to another context for which κ holds. A CE $\oplus\kappa$ holds for a context if that context has a child context for which κ holds. A CE $✧\kappa$ holds for a context if there exists somewhere in that context a sub-context for which κ holds. The operator $✧$ is called *somewhere modality* while $\oplus$ is aka *spatial next modality*.

Example 3.2 *We now give some examples of predicates that can be used to specify common context properties such as the location of the user, with whom the user is and what resources are nearby. In these sample predicates we take the view that a process is evaluated by the immediate ambient λ say that contains it.*

- $\texttt{has}(n) \ \widehat{=}\ \oplus\,(\bullet \mid n[\mathbf{True}] \mid \mathbf{True})$ *: holds if ambient λ contains an ambient named n*
- $\texttt{at}(n) \ \widehat{=}\ n[\oplus(\bullet \mid \mathbf{True})] \mid \mathbf{True}$ *: holds if ambient λ is located at an ambient named n*
- $\texttt{with}(n) \ \widehat{=}\ n[\mathbf{True}] \mid \oplus(\bullet \mid \mathbf{True})$ *: holds if ambient λ is (co-located) with an ambient named n*

4 Formalisation

This section presents the formalisation of the IMN service, in which we present the specification of the IS, the ISC and all their associated ambients. But first, we give the naming convention which we are going to use to model the infostation-based mLearning system.

4.1 Notations

Table 3 presents naming conventions to differentiate between variables and constants; a variable name begins with a lowercase letter while a constant name begins with a number or uppercase letter.

Table 3. Variables and constants

Notations	Description	Values
Variables		
uid	a user's ID	305, 306
sid	a user's ID	305, 306
rid	a user's ID	305, 306
$dtype$	a user's device type	Phone, PDA, PC
$sdtype$	a user's device type	Phone, PDA, PC
$rdtype$	a user's device type	Phone, PDA, PC
$aname$	an ambient's name	Phone306, PC305
$saname$	an ambient's name	Phone306, PC305
$raname$	an ambient's name	Phone306, PC305
$reply$	a reply to a request	DENIED
ack	acknowledgment	ACK
Constants		
ISC	the InfoStation Centre	
IS_i	an infostation	$IS_1, IS2$
$Phone$	a phone	
PDA	a PDA	
PC	a PC	
ACK	an acknowledgment	
$DENIED$	request denied	

4.2 Modelling the Infostation-based mLearning system

The infostation-based mLearning system consists mainly of one central ISC, multiple ISs and multiple user devices. Each component of the system is modelled as an ambient in CCA. For example, the ISC is modelled as an ambient named ISC, and user named 305 carrying a PC is modelled as an ambient named $PC305$. The ISC ambient runs in parallel with the IS ambients, and all the users' devices within the range of an IS are child ambients of that IS ambient and siblings to each other, as illustrated in Figure 3 below.

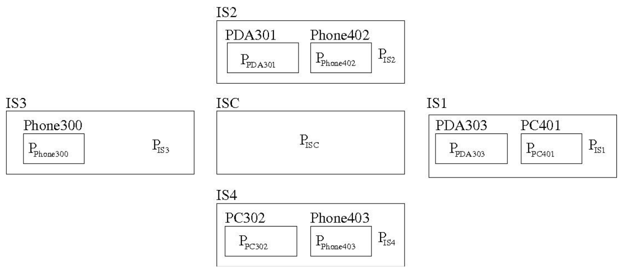


Figure 3. A model of the infostation-based mLearning system

where each P_x is a process modelling the behaviour of the corresponding ambient x. Now we give the formal specification of the ISs and the ISC.

4.2.1 Infostation

A model of an IS is depicted in Figure 4. This model is a simple one in which it contains the *IMNreq* ambient only, for in this paper we specify the IMN service. However, the full model of an IS contains the ambients that are responsible for other services, and the ambient that models the cache of that IS.

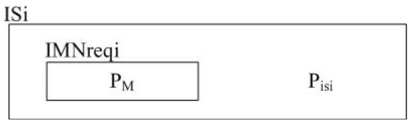


Figure 4. A model of an infostation

IMNreq It is the ambient responsible for handling IMN requests made by users' devices. We assume that only text messages can be send via this service, so that every device in our system is capable of sending and receiving them. This ambient receives a request from a user device and forwards it to the IS_i ambient, as follows:

$$\begin{array}{l} ! :: (msg, rid, raname, sid, saname). \\ IS_i \uparrow \langle msg, rid, raname, sid, saname\rangle.\mathbf{0} \end{array} \tag{2}$$

The *IMNreq* ambient receive a request from IS to deliver a message to a recipient, this process is modelled as follows, where it receives the request and immediately forwards it.

$$!IS_i \uparrow (msg, sid, raname).raname :: \langle msg, sid\rangle.\mathbf{0} \tag{3}$$

The whole behaviour of the $IMNreq$ ambient is:

$$P_{M_i} \mathrel{\widehat{=}} \text{Eq. } (2) \mid \text{Eq. } (3) \tag{4}$$

ISi An IS receives a request from the *IMNreq* ambient and checks to see whether the recipient is inside its range, then it acts. This piece of the model (Eq.5) represents the intelligence part of the IMN service, in which the infostation checks the location of the recipient first rather than forwarding the request to the ISC.

$$\begin{array}{l} !IMNreq\downarrow(msg, rid, raname, sid, saname). \\ \left(\begin{array}{l} \mathtt{has}(raname)?\ A1 \mid \\ \neg(\mathtt{has}(raname))?\ A2 \end{array}\right) \end{array} \tag{5}$$

Where A1 is the act in case the recipient is not in the range of the same IS, as follows:

$$A1 \mathrel{\widehat{=}} ISC :: \langle msg, rid, raname, sid, saname\rangle.\mathbf{0}$$

And A2 is the act in case the recipient is in the range of the same IS, where the IS first sends two requests sequentially to the ISC to check whether the sender or the recipient are using the mTest service; if yes (i.e. either of them is taking a mTest), it sends a DENIED message back to the *IMNreq* ambient, if no, it sends the message to the recipient, as follows:

$$\begin{array}{l} A2 \mathrel{\widehat{=}} ISC :: \langle Utest, sid, saname\rangle.ISC :: (reply, saname). \\ \left(\begin{array}{l} (reply = DENIED)?\ IMNreq\downarrow\langle reply, rid, saname\rangle.\mathbf{0} \mid \\ \neg(reply = DENIED)?\ ISC :: \langle Utest, rid, raname\rangle. \\ ISC :: (reply, raname). \\ \left(\begin{array}{l} (reply = DENIED)?\ IMNreq\downarrow\langle reply, rid, saname\rangle.\mathbf{0} \mid \\ \neg(reply = DENIED)?\ IMNreq\downarrow\langle msg, sid, raname\rangle.\mathbf{0} \end{array}\right) \end{array}\right) \end{array}$$

Eventually, the IS receives a request from the ISC to deliver a reply to a sender, or a message to a recipient, the infostation just forwards the received parameters to the *IMNreq* ambient. This process is modelled as follows, where the parameters (a, b, c) are the alternatives of the variables $(reply, rid, saname)$ or $(msg, sid, raname)$.

$$!ISC :: (a, b, c).IMNreq\downarrow\langle a, b, c\rangle.\mathbf{0} \tag{6}$$

Also, the IS may receive a reply from the *IMNreq* and forwards it to the ISC, as follows:

$$!IMNreq\downarrow(ack, sid, raname).ISC :: \langle ack, sid, raname\rangle.\mathbf{0} \tag{7}$$

So the whole behaviour of the IS is as follows:

$$P_{IS_i} \mathrel{\widehat{=}} \text{Eq. } (5) \mid \text{Eq. } (6) \mid \text{Eq. } (7) \tag{8}$$

4.2.2 Infostation Centre

A model of the ISC is depicted in Figure 5. This model contains the ambients of the users' accounts. However, the full model of the ISC contains the services profiles (i.e. database for lecture/test materials).

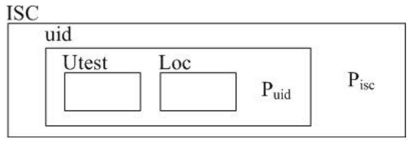


Figure 5. A model of the Infostation Centre

Users' accounts An ambient modelling a user's account which contains two ambients: Loc and $Utest$. Each of these two ambients models a persistent memory cell (as in Eq. (1)) which stores, at any time, the current location of that user (for the former) or a Boolean indicating whether that user is taking a mTest or not (for the latter). The ISC requests the value of any of these cells by sending the name Loc or $Utest$ to the user's account ambient which then can *get* the value of the corresponding child ambient as follows, where the parameter x is the corresponding child ambient name:

$$!ISC\uparrow(x).x\downarrow\langle\rangle.x\downarrow(y).\,ISC\uparrow\langle y\rangle.\mathbf{0} \tag{9}$$

It can also *put* a value in any of its child ambients as follows, where the parameter x is the corresponding child ambient name and the parameter n is that value:

$$!ISC\uparrow(x,n).x\downarrow\langle n\rangle.x\downarrow().ISC\uparrow\langle x, ACK\rangle.\mathbf{0} \tag{10}$$

So the whole behaviour of a user's account ambient named uid is specified as:

$$P_{uid} \;\widehat{=}\; \text{Eq. } (9) \mid \text{Eq. } (10).$$

ISC We now formalise the behaviour of the ISC concerning the IMN service. When the ISC receives an IMN request from an IS, it forwards the request to the corresponding child ambient (i.e. an IS) after checking that the sender and the recipient are currently *not* using the mTest service. This process is modelled as follows:

$$\begin{array}{l} !ISi :: (msg, rid, raname, sid, saname).\; sid\downarrow\langle Utest\rangle.\; sid\downarrow(y_{sid}). \\ (I1 \mid I2) \end{array} \tag{11}$$

Where

$$\begin{array}{l} I1 \;\widehat{=}\; (y_{sid}=1)?\; sid\downarrow\langle Loc\rangle.\,\mathbf{0} \mid sid\downarrow(z_{sid}). \\ z_{sid} :: \langle DENIED, raname\rangle.\,\mathbf{0} \end{array}$$

and

$$\begin{array}{l} I2 \;\widehat{=}\; (y_{sid}=0)?\; rid\downarrow\langle Utest\rangle.\,\mathbf{0} \mid rid\downarrow(y_{rid}). \\ \left(\begin{array}{l} (y_{rid}=1)?\; sid\downarrow\langle Loc\rangle.\,\mathbf{0} \mid sid\downarrow(z_{sid}). \\ z_{sid} :: \langle DENIED, raname\rangle.\mathbf{0} \mid \\ (y_{rid}=0)?\; rid\downarrow\langle Loc\rangle.\,\mathbf{0} \mid rid\downarrow(z_{rid}). \\ z_{rid} :: \langle msg, sid, raname\rangle.\mathbf{0} \mid z_{rid} :: (ACK, sid, raname). \\ sid\downarrow\langle Loc\rangle.\,\mathbf{0} \mid sid\downarrow(z_{sid}).\; z_{sid} :: \langle ACK, sid, raname\rangle.\mathbf{0} \end{array} \right) \end{array}$$

As mentioned earlier in Eq.(5) that the recipient of the IMN service may be located in the same IS as of the sender, in this case, the ISC receives a request from an IS to *only* validate the sender and the recipient (i.e. not using the mTest service), as follows:

$$\begin{array}{l} !IS_i :: (Utest, uid, aname).uid\downarrow\langle Utest\rangle.uid\downarrow(y). \\ \left(\begin{array}{l} (y=1)?\; IS_i :: \langle DENIED, aname\rangle.\mathbf{0} \mid \\ (y=0)?\; IS_i :: \langle OK, aname\rangle.\mathbf{0} \end{array} \right) \end{array} \tag{12}$$

However, the whole behaviour of the ISC is modelled as:

$$P_{ISC} \;\widehat{=}\; \text{Eq. } (\mathbf{??}) \mid \text{Eq. } (11) \mid \text{Eq. } (12) \tag{13}$$

5 Validation

In this section, we validate a liveness property of the IMN service that has been specified in the previous section. To validate the property, we will use the CCA execution environment showed in Figure 6. To break down the component of the execution environment, the user is free to use any text editor to write a specification, the specification will then be checked syntactically by the parser which has been developed along with the interpreter using JavaCC (Java Compiler Compiler). The output of the execution will then be showed using the console, the animator has been developed as a visual aid to animate CCA processes, however, we are not going to use it in this paper.

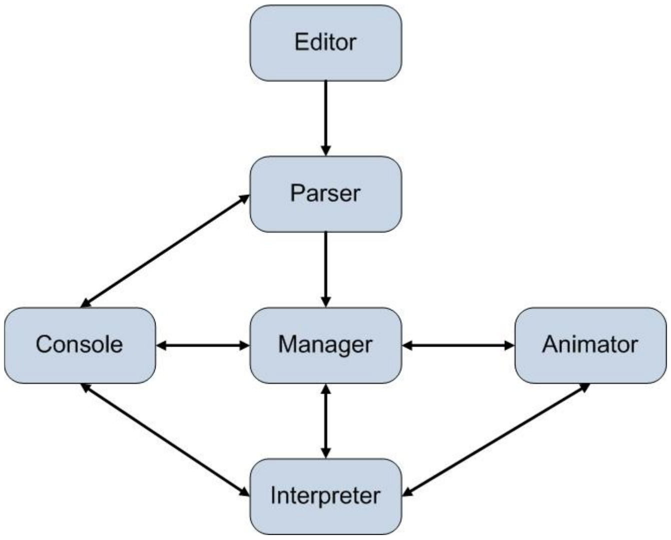


Figure 6. The architecture of the execution environment of CCA

The validation process is done by executing the previously specified specification of the IS, ISC in parallel with users behaviour. The syntax used in the CCA execution environment is the Programming Language for the Calculus of Context-aware Ambient (ccaPL in short), which is a machine readable format of the syntax of CCA. The CCA compiler is based on the reduction semantics of CCA.

In this section, we, will validate a liveness property of the mLearning system using two user's behaviours for illustration. Subsequently, this process shows an example that using our approach, a whole context-aware system can be validated by (i) writing formal specification of a system; (ii) creating scenarios corresponding to different users' behaviour and (iii) Simulating the users' behaviour against the system specification.

5.1 Syntax of ccaPL

This section presents the syntax of ccaPL. The syntax of ccaPL is almost identical to the syntax of CCA; however, there are some symbols in CCA syntax that are not machine readable, so we present their equivalent in ccaPL syntax.

Table 4. Processes of ccaPL

Processes		
CCA	ccaPL	**Description**
$\mathbf{0}$	$\mathbf{0}$	inactivity
$n[p]$	$n[p]$	ambient
$P\|Q$	$P\|Q$	parallel composition
$!P$	$!P$	replication
$(vn)P$	new nP	restriction
$\kappa?M.P$	$< \kappa > M.P$	context-guarded capability
$x \triangleright (\tilde{y}).P$	proc $x(\tilde{y}).P$	process abstraction x
P	P	brackets

Table 5. Capabilities of ccaPL

Capabilities		
CCA	ccaPL	**Description**
in n	in n	move into the ambient n
out	*out*	move out
del n	del n	delete ambient n
$\alpha\ x\langle\tilde{z}\rangle$	$\alpha\ x(\tilde{z})$	call to the process abstraction x
$\alpha\ (\tilde{y})$	α recv$(\tilde{y})$	receive list of messages $\tilde{y}$ from α
$\alpha\ \langle\tilde{y}\rangle$	α send$(\tilde{z})$	send list of messages $\tilde{z}$ to α

5.2 Executing the IMN Service

In this section, we will validate an important *liveness* property of the mLearning system. As proposed by Lamport, a liveness property which states that '*something good will happen, eventually*'. In the light of this definition, we will validate a property regarding the IMN service which is, when a user sends a message to another user, that user will eventually receive the message, provided that the other user does not become infinitely unavailable after the message is sent. However, both users are in the same IS.

Theorem 5.1 *When a user sends a message to another user, the message will eventually be received, provided that the recipient stays long enough within an IS.*

Table 6. Location of ccaPL

Location		
CCA	ccaPL	**Description**
$\uparrow$	@	any parent
$n\uparrow$	n@	parent n
$\downarrow$	#	any child
$n\downarrow$	n#	child n
::	::	any sibling
n ::	n ::	sibling n
ϵ	ϵ	locally

The proof of this theorem is based on the ccaPL execution environment. In this proof, we assume that the sender is user 305 who is using a PC, and the receiver is user 306 who is using a phone device. However, with both users are mobile, two scenarios apply:

1. The sender and the recipient are in the same IS, say $IS1$;
2. The sender and recipient are in different ISs, say $IS1$ and $IS2$.

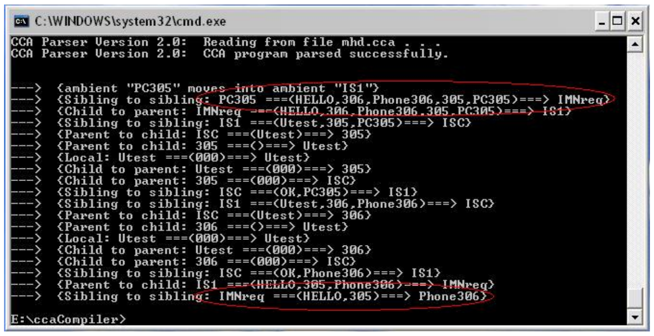
Now, in Case 1, the sender's behaviour can be modelled as follows:

```
PC305[IMNreq :: send(HELLO, 306, Phone306, 305, PC305).0]
```

and the recipient's behaviour is modelled as follows:

```
Phone306[IMNreq :: recv(msg, sid).0]
```

Given that both users are in the same IS, we execute the behaviours of both users, IS and ISC in parallel using the execution environment of ccaPL. Figure 7 shows the execution of the above scenario (users in the same IS).



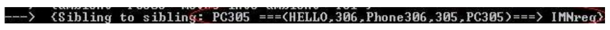
```
C:\WINDOWS\system32\cmd.exe
CCA Parser Version 2.0:  Reading from file mhd.cca . . .
CCA Parser Version 2.0:  CCA program parsed successfully.


--->  {ambient "PC305" moves into ambient "IS1"}
--->  {Sibling to sibling: PC305 ===(HELLO,306,Phone306,305,PC305)===> IMNreq}
--->  {Child to parent: IMNreq ===(HELLO,306,Phone306,305,PC305)===> IS1}
--->  {Sibling to sibling: IS1 ===(Utest,305,PC305)===> ISC}
--->  {Parent to child: ISC ===(Utest)===> 305}
--->  {Parent to child: 305 ===()===> Utest}
--->  {Local: Utest ===(000)===> Utest}
--->  {Child to parent: Utest ===(000)===> 305}
--->  {Child to parent: 305 ===(000)===> ISC}
--->  {Sibling to sibling: ISC ===(OK,PC305)===> IS1}
--->  {Sibling to sibling: IS1 ===(Utest,306,Phone306)===> ISC}
--->  {Parent to child: ISC ===(Utest)===> 306}
--->  {Parent to child: 306 ===()===> Utest}
--->  {Local: Utest ===(000)===> Utest}
--->  {Child to parent: Utest ===(000)===> 306}
--->  {Child to parent: 306 ===(000)===> ISC}
--->  {Sibling to sibling: ISC ===(OK,Phone306)===> IS1}
--->  {Parent to child: IS1 ===(HELLO,305,Phone306)===> IMNreq}
--->  {Sibling to sibling: IMNreq ===(HELLO,305)===> Phone306}

E:\ccaCompiler>
```

Figure 7. Execution of IMN service: same IS

where the single line:



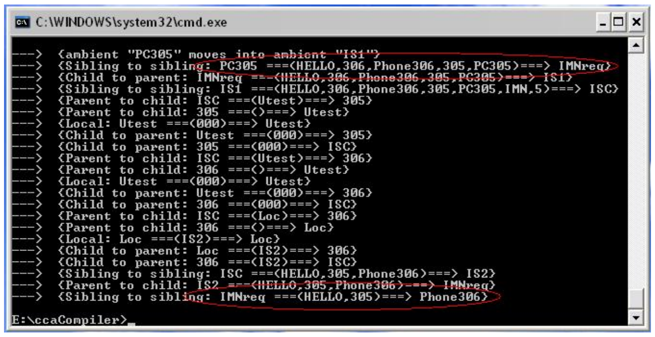
```
--->  {Sibling to sibling: PC305 ===(HELLO,306,Phone306,305,PC305)===> IMNreq}
```

which is:

```
--> <sibling to sibling: PC305===(HELLO, 306, Phone306, 305, PC305)===> IMNreq>
```

means that a sibling has communicated with another sibling; and the process is: *PC305* has sent the parameters $(\mathit{HELLO}, \mathit{306}, \mathit{Phone}, \mathit{305}, \mathit{PC})$ to the ambient *IMNreq*. However, the execution shows the transitions that have happened, sequentially. The execution shows in the last transition that the *IMNreq* ambient inside IS1 has sent the message '*HELLO*' to the ambient *Phone306* which is the recipient. And that proofs the first scenario of the property.

For the second scenario, Figure 8 shows the execution of the IMN service, in which both users have the same behaviour as the above scenario; however, they are in different ISs: *IS1* and *IS2*.



```
C:\WINDOWS\system32\cmd.exe
--->  <ambient "PC305" moves into ambient "IS1">
--->  <Sibling to sibling: PC305 ===<HELLO,306,Phone306,305,PC305>===> IMNreq>
--->  <Child to parent: IMNreq ===<HELLO,306,Phone306,305,PC305>===> IS1>
--->  <Sibling to sibling: IS1 ===<HELLO,306,Phone306,305,PC305,IMN,5>===> ISC>
--->  <Parent to child: ISC ===<Utest>===> 305>
--->  <Parent to child: 305 ===<>===> Utest>
--->  <Local: Utest ===<000>===> Utest>
--->  <Child to parent: Utest ===<000>===> 305>
--->  <Child to parent: 305 ===<000>===> ISC>
--->  <Parent to child: ISC ===<Utest>===> 306>
--->  <Parent to child: 306 ===<>===> Utest>
--->  <Local: Utest ===<000>===> Utest>
--->  <Child to parent: Utest ===<000>===> 306>
--->  <Child to parent: 306 ===<000>===> ISC>
--->  <Parent to child: ISC ===<Loc>===> 306>
--->  <Parent to child: 306 ===<>===> Loc>
--->  <Local: Loc ===<IS2>===> Loc>
--->  <Child to parent: Loc ===<IS2>===> 306>
--->  <Child to parent: 306 ===<IS2>===> ISC>
--->  <Sibling to sibling: ISC ===<HELLO,305,Phone306>===> IS2>
--->  <Parent to child: IS2 ===<HELLO,305,Phone306>===> IMNreq>
--->  <Sibling to sibling: IMNreq ===<HELLO,305>===> Phone306>

E:\ccaCompiler>_
```

Figure 8. Execution of IMN service: different ISs

The execution shows in the last transition that the *IMNreq* ambient inside IS2 has sent the message '*HELLO*' to the ambient *Phone306* which is the recipient. And that proofs the second part of the property.

6 Related Work

In a previous published paper [1], we have introduced the specification of the AAA and mLecture services, and we proved an important liveness property for the mLearning system using the reduction semantics of CCA.

A UML specification of an infostation-based mLearning system was proposed in [4]. Although specification with UML provides a number of advantages such as code generation and other system analyses using assistant tools; however, the lack of support for formal reasoning is one of its limitations for the design of critical systems. Through a UML diagram, a specific behaviour of a system can only be observed.

7 Conclusion and Future Work

We introduced a real-world case study of an infostation-based mLearning system, with emphasis on the IMN service. Then we introduced CCA which was proposed as suitable process calculus to model mobile systems that are context-aware. Using the notation of CCA, we specified the mLearning system with regards to the IMN service, and using the execution environment of CCA, we executed a couple of scenarios related to that service. This case study is yet another demonstration of the expressiveness and pragmatics of CCA.

We validated a liveness property of the mLearning system for illustration. However, we can conclude that using our approach, a whole context-aware system can be validated by (i) writing the formal specification of a system; (ii) creating scenarios corresponding to different users' behaviour and finally (iii) simulating the users' behaviour against the system specification.

In a future work, we will extend the formalisation of the mLearning system to encompass other services. In regards to the execution environment of ccaPL, an animation tool of CCA specification has been developed. With the parser, interpreter and animator, the CCA execution environment is the only complete executable specification of context-aware mobile applications.

References

[1] M. Al-Sammarraie, F. Siewe, and H. Zedan. Formalising policies of a mlearning system using cca. In *Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems*, CASEMANS '10, pages 2:11–2:19, New York, NY, USA, 2010. ACM.

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

[3] R. H. Frenkiel and T. Imielinski. Infostations: The joy of 'many-time, many-where' communications. Technical report, WINLAB, 1996.

[4] I. Ganchev, S. Stojanov, M. O'Droma, and D. Meere. An infostation-based multi-agent system supporting intelligent mobile services across a university campus. *Journal of Computers*, 2(3), 2007.

[5] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

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

[7] D. S. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2003.