Formalising Policies of a mLearning System using CCA

Mohammed Al-Sammarraie
Software Technology Research Laboratory
De Montfort University
Leicester, UK
mhd@dmu.ac.uk

François Siewe
Software Technology Research Laboratory
De Montfort University
Leicester, UK
fsiewe@dmu.ac.uk

Hussein Zedan
Software Technology Research Laboratory
De Montfort University
Leicester, UK
hzedan@dmu.ac.uk

ABSTRACT

The Calculus of Context-aware Ambients (CCA in short) has been proposed as a suitable notation for modelling mobile applications that are context-aware. This paper considers a real-world case study of an infostation-based mLearning system which enables mobile devices such as cellular phones, laptops and personal digital assistants to communicate to each other and to access a number of mlearning services within a university campus. Such a dynamic system must enforce complex policies to cope with mobility and context-awareness. We show how these policies can be formalised and verified using CCA. In particular an important liveness property of the mLearning system is proved using the reduction semantics of CCA.

Categories and Subject Descriptors

F.1.1 [**Computation by abstract devices**]: Modes of computation—*Interactive and reactive computation*; H.1.1 [**Models and principles**]: Systems and information theory—*General systems theory*

General Terms

Language, theory, verification

Keywords

Context-Awareness, mLearning, InfoStation, ambient, process calculus

1. INTRODUCTION

The world is increasingly becoming a larger network fully enhanced with a multitude of sensors and smart agents that have the ability to detect, sense and adapt to changes in people, devices, locations, applications and events which determine the everyday social behaviour. The topology of such a network is not fixed but changes dynamically in an unpredictable manner. The design and the verification of such a complex system poses many challenges faced by the ubiquitous computing community. Recently, the Calculus of Context-aware Ambients (CCA in short) [13] has been proposed as a suitable mathematical notation for modelling these kind of systems at a reasonable level of abstraction. Unlike other process calculi (such as [3, 9, 6, 8]), *context-awareness* and *mobility* are first-class citizens in CCA.

As eLearning (including mobile learning, aka *mLearning*) is becoming a real viable alternative educational approach to the face-to-face traditional learning, it is important that such a system be modelled and its properties verified to ensure the provision of better QoS and user experience. So this paper investigates the modelling and verification in CCA of a real-world case study of a context-aware mLearning system. The system is based on the infostation paradigm which is an infrastructural system concept proposed by [4] to provide the 'many-time, many-where' wireless data services. The considered case study is a mLearning system which enables mobile devices such as cellular phones, laptops and personal digital assistants to access a number of mlearning services (e.g. mLecture, mTutorial, mTest and Intelligent Message Notification service) within a university campus. Such a dynamic system must enforce complex policies to cope with mobility and context-awareness.

This paper presents how this mLearning system can be formalised and verified in CCA in a natural manner. Our contributions are summarised as follows:

- Elicitation of the policies of two services of the mLearning system (Sect. 2);
- Formalisation of these policies in CCA (Sect. 4);
- Verification of an important liveness property of the mLearning system (Sect. 5) using the reduction semantics of CCA.

2. INFOSTATION-BASED MLEARNING SYSTEM

eLearning is becoming an authentic possible alternative approach as the technologies regarding that area are developing so fast, and there is a recognisable growth of a great variety of wide-band telecommunication delivery technologies. The infostation paradigm first proposed by Frenkiel et al. [4] and used in [5] to devise an infostation-based mlearning system which allows mobile devices such as cellular phones, laptops and personal digital assistants (PDAs) to communicate to each other and to a number of services within a university campus. This mLearning system provides a number of services among which are: mLecture, mTutorial, mTest and communication services (private chat, intelligent message notification and phone calls). This section presents the architecture of the infostation-based mLearning system and describes the policies of the mLecture service.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CASEMANS '10, Sep 26, 2010, Copenhagen, Denmark.
Copyright 2010 ACM 978-1-4503-0213-5...$10.00.

2.1 mServices

This section introduces at glance each of the mServices provided by the infostation-based mLearning system.

- *AAA*: in order for any user to use any mService in the system, the user device should be registered. The AAA service (Authentication, Authorisation and Accounting) allows the users to register their devices with the system to gain the ability of using the mLearning services offered by the system.

- *mLecture*: this service allows the users to gain access to the lecture material through their mobile devices. The users can request a specific lecture, which is adapted according to the capabilities of the user devices and then delivered to their mobile devices.

- *mTest*: this service is crucial to the learning process. The mTest service allows the users to gain access to test materials that provide means of an evaluating process. A user can request, like the mLecture service, a specific test, which is also adapted to the capabilities of the user device then delivered to the user mobile device. The mTest service may only runs individually on a user device and unaccompanied with any other service whatsoever.

- *mTutorial*: this service allows the users to gain access to a self-assessment test. It is a combination between the mLecture and the mTest services. A user can request a self-assessment test in a similar way as requesting a mLecture. After the user submits their answers, he receives a feedback on his performance and the correct answers to the questions he got wrong.

- *Intelligent Message Notification (IMN in short)*: this service allows the users to communicate with each other by exchanging messages via their mobile devices.

- *VoIP*: this service allows the users to communicate with each other via phone calls throughout the infostation-based mLearning system.

The following section describes the architecture of the infostation-based mLearning system.

2.2 Architecture

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

The first tier encompasses users' mobile devices (e.g. Laptop, Phone and PDA). 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 around a university campus. These ISs enhance the operation of supplying the users with a mobile access to the mServices, through high-speed connections [5]. Infostations are usually deployed irregularly (i.e. geographically) and are equipped to support different communication protocols (Figure 2), so as to provide a successful access to a wide range of mobile devices. The third tier is the InfoStation Centre (ISC in short), the core of the architecture. Its main function is to control the ISs, update and synchronise information across the system. It also manages the users' and services' profiles.

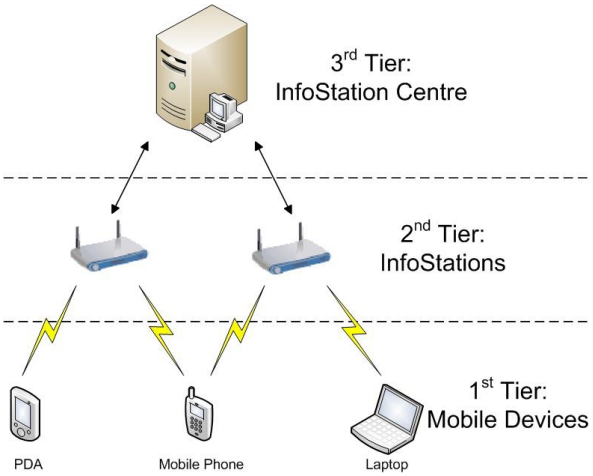


Figure 1: The three tiers architecture of the InfoStation-based network [5]

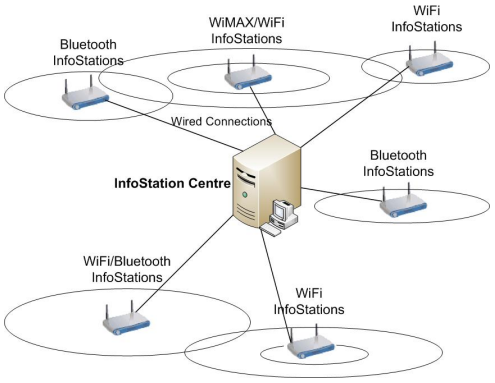


Figure 2: The InfoStation-based network architecture

2.3 Policies

The ISC provides the User Authentication, Authorisation and Accounting (AAA) service which identifies each mobile user and provides him with a list of services the user is authorised to access. This service is regulated by the following policies:

- When a user enters within the range of an IS, the intelligent agent (PA) of the user's device and the IS mutually discover each other. The PA sends a request to the IS for user Authorisation, Authentication and Accounting (AAA). This request also includes a description of the mobile device currently being used and any updates of user profile and user service profile.

- The IS forwards this AAA request to the ISC along with the profile updates. If the user is successfully authenticated and authorised to utilise the services by the AAA module within the ISC, a new account record is created for the user and a positive acknowledgement is sent back to the IS. Then the IS compiles a list of applicable services and sends this to the PA along with the acknowledgement. The PA displays the information regarding these services to the user who then makes a request for the service he wishes to use.

If the user chooses the mLecture service, then the following policies of the mLecture service apply:

- The PA forwards the mLecture service request to the InfoStation, which instantiates the service. If the IS is unable to satisfy fully the user service request it is forwarded to the ISC, which is better equipped to deal with it. In either case, the lecture is adapted and customised to suit the capabilities of the user devices and the user own preferences, and then delivered to their mobile devices.
- During the execution of the service, the user is free to move into a different infostation, to switch between devices or to do both.
- A user cannot use the mLecture and mTest services simultaneously. The mTest service should operate unaccompanied at all occasions.

The following section presents a process calculus which will be used to model and verify properties of these policies.

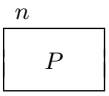
3. OVERVIEW OF CCA

CCA [13] 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) [3] and provides primitives for modelling mobility and context-awareness. An ambient is an abstraction of a bounded place where computation happens. It can be mobile, non-mobile, 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 [13] for the formal semantics of CCA. Like in the π-calculus [9, 11], 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

P, Q	$::=$	$\mathbf{0} \mid P\|Q \mid n[P] \mid !P \mid \kappa?M.P \mid x \triangleright (\tilde{y}).P$
M	$::=$	$\texttt{in}\ n \mid \texttt{out} \mid \alpha\ x\langle\tilde{y}\rangle \mid \alpha\ (\tilde{y}) \mid \alpha\ \langle\tilde{y}\rangle \mid \texttt{del}\ n$
α	$::=$	$\uparrow \mid n\uparrow \mid \downarrow \mid n\downarrow \mid\ ::\ \mid n ::\ \mid \epsilon$
κ	$::=$	$\mathbf{True} \mid\ \bullet\ \mid n = m \mid \neg\kappa \mid \kappa_1 \mid \kappa_2 \mid \kappa_1 \wedge \kappa_2 \mid \oplus\kappa \mid \diamond\!\!\!\!-\kappa \mid \exists x.\kappa$

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 [3]. An ambient that performs the capability $\texttt{in}\ n$ moves into the sibling ambient n. The capability out moves the ambient that performs it out of that ambient's parent. The capability $\texttt{del}\ n$ deletes an ambient of the form $n[\mathbf{0}]$ situated at the same level as that capability, i.e. the process $\texttt{del}\ n.P \mid n[\mathbf{0}]$ 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:*

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

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

Table 2: Syntax of contexts

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

of contexts are called context expressions (CEs in short).

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 $\mathtt{new}(n, \kappa)$ 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 $\diamond\kappa$ holds for a context if there exists somewhere in that context a sub-context for which κ holds. The operator $\diamond$ 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.*

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

4. FORMALISATION

This section presents the formalisation of the policies of the infostation-based context-aware mLearning system using CCA. We first introduce some naming conventions (sect. 4.1) which are used in the specification of the system. Then we give the specification of two mServices which are AAA and the $mLecture$ services (sect. 4.2).

4.1 Notations

The following naming conventions are used to differentiate between variables' names and constants. A variable name begins with a lowercase letter while a constant begins with a number or a uppercase letter. The list of the constant names that are used in the formalisation process is given in Table 3. And the list of variable names is given in Table 4.

Table 3: Constants

Constants	
Notations	**Description**
ISC	the InfoStation Centre
IS_i	the InfoStation i; e.g. IS_1
$Phone$	a phone
PDA	a PDA
PC	a PC
$SLIST$	list of mServices
ACK	an acknowledgement
NULL	empty message
DENIED	request denied

Table 4: Variables

Variables		
Notations	**Description**	**Values**
uid	a user's ID	301, 302, 303
$dtype$	a user's device type	Phone, PDA, PC
$aname$	an ambient's name	Phone301, PC303
$lect$	a lecture's ID	Lect001
$reply$	a reply to a request	OK, DENIED, content
$content$	lecture's content	CONTENT
$slist$	list of services	SLIST
ack	acknowledgement	ACK

4.2 A Model of the InfoStation-based mLearning System

The system consists mainly of one central ISC, multiple ISs and multiple user devices. Each component of the system, as depicted in Figure 1, is modelled as an ambient in CCA. That is, the ISC, each IS and each user device is modelled as an ambient. In particular, a device, PC say, being used by a user, 303 say, is modelled by an ambient named $PC303$. The ISC ambient runs in parallel with the IS ambients, and all the user devices within the range of an IS are child ambients of that IS ambient as illustrated graphically in Figure 3.

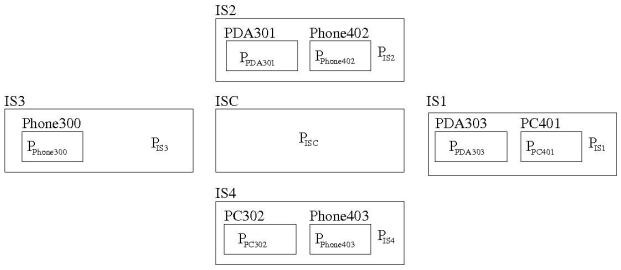


Figure 3: A model of the infostation-based mLearning system

This is textually represented by the following CCA process:

$$\begin{aligned} ISC[P_{ISC}] \quad &| \quad IS1[PDA303[P_{PDA303}] \mid PC401[P_{PC401}] \\ &\quad \mid P_{IS1}] \\ &| \quad IS2[PDA301[P_{PDA301}] \\ &\quad \mid Phone402[P_{Phone402}] \mid P_{IS2}] \\ &| \quad IS3[Phone300[P_{Phone300}] \mid P_{IS3}] \\ &\quad \mid IS4[Phone403[P_{Phone403}] \mid PC302[P_{PC302}] \\ &\quad \mid P_{IS4}] \end{aligned} \tag{2}$$

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

4.2.1 InfoStation

An abstract model of an infostation IS_i (for some integer i) is shown in Figure 4 where the main components are the AAA request ambient $AAAreq_i$, the lecture ambient $Lectreq_i$ and the cache ambient $Cache_i$.

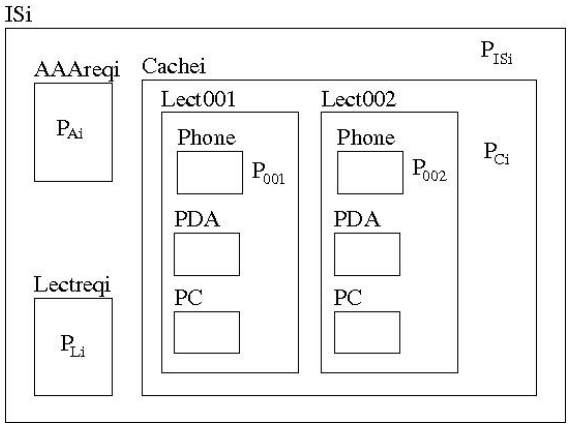


Figure 4: A model of an Infostation

The InfoStation is a parent to the inside ambients which are siblings to each other. The specification of each of these ambients is as follows:

$AAAreq_i$. This ambient is responsible for handling AAA requests sent by user devices willing to register with the InfoStation IS_i. The $AAAreq_i$ ambient receives a request from a device and, immediately, forwards it to the InfoStation, then receives a reply from the InfoStation and again, forwards it to the user's device. This behaviour is modelled by the following process:

$$\begin{aligned} P_{A_i} \;\widehat{=}\; &!::(uid, dtype, aname).IS_i \uparrow \langle uid, dtype, aname\rangle. \\ &IS_i \uparrow (ack, aname, slist).aname :: \langle ack, slist\rangle.\mathbf{0} \end{aligned}$$

where uid is the user ID, $dtype$ is the device type and $aname$ is the name of the ambient sending the request.

The InfoStation accordingly receives a request from the $AAAreq_i$ ambient, forwards it to the InfoStation Centre, and after receiving the reply from the InfoStation Centre it forwards it to the $AAAreq_i$ ambient. This behaviour is modelled as:

$$\left(\begin{array}{l} !AAAreq_i{\downarrow}(uid, dtype, aname).ISC :: \langle AAAreq, uid, \\ dtype, aname, IS_i\rangle.\mathbf{0} \mid \\ \\ !ISC :: (ack, aname, slist).(\mathtt{has}(aname))? \\ AAAreqi{\downarrow}\langle ack, aname, slist\rangle.\mathbf{0} \end{array}\right) \tag{3}$$

$Lectreq_i$. This ambient handles all the mLecture service requests sent by the user devices. It receives a lecture request from a user device and forwards it to the infostation IS_i, i.e.

$$\begin{aligned} &!::(lectid, uid, dtype, aname). \\ &IS_i \uparrow \langle lectid, uid, dtype, aname\rangle.\mathbf{0} \end{aligned} \tag{4}$$

Then it gets the reply from that infostation and forwards it to the user device which initiated the request, i.e.

$$!IS_i \uparrow (lectid, reply, aname).aname :: \langle lectid, reply\rangle.\mathbf{0} \tag{5}$$

So the whole behaviour of the $Lectreq_i$ ambient is

$$P_{L_i} \;\widehat{=}\; \text{Eq. } (4) \mid \text{Eq. } (5) \tag{6}$$

We will show how the InfoStation handles a request from the $Lectreq_i$ ambient after we have specified the $Cache_i$ ambient.

$Cache_i$. This is the ambient where the InfoStation stores copies of requested lectures for future rapid access. It models a cache memory. A lecture is stored as an ambient (named after that lecture's id) which contains three persistent memory cells (see Example 3.1), each containing a version of the lecture suitable to a specific type of device (phone, PDA or PC). When an InfoStation receives a mLecture service request from a device, it checks for the requested material in its cache first rather than getting it from the InfoStation Centre directly. The process of checking the availability of a lecture inside the cache is done by sending a request to the $Cache_i$ ambient which then checks whether it has the ambient of the requested lecture or not. If the requested lecture is available the cache ambient retrieves it and sends it back to the InfoStation, otherwise, it replies immediately to the InfoStation that this lecture does not exist. The behaviour of the $Cache_i$ ambient is modelled by the following process:

$$P_{c_i} \;\widehat{=}\; !\uparrow(lectid, uid, dtype, aname).\left(\begin{array}{l}\texttt{has}(lectid)?lectid\downarrow\langle dtype, aname\rangle.\\ lectid\downarrow(reply, aname).\\ \uparrow\langle lectid, uid, dtype, reply, aname\rangle.\mathbf{0}\mid\\ \\ \neg\texttt{has}(lectid)?\uparrow\langle lectid, uid, dtype,\\ NULL, aname\rangle.\uparrow(lectid, content, dtype).\\ lectid\downarrow\langle content, dtype\rangle.lectid\downarrow(ack).\uparrow\langle ack\rangle.\mathbf{0}\end{array}\right) \tag{7}$$

The behaviour of each lecture ambient (named after the lecture's id $lectid$) in the cache is modelled by the following process:

$$P_{lectid} \;\widehat{=}\; !\uparrow(dtype, aname).\left(\begin{array}{l}\texttt{has}(dtype)?dtype\downarrow\langle\rangle.\ dtype\downarrow(reply).\\ \uparrow\langle reply, aname\rangle.\ \mathbf{0}\mid\\ \\ \neg\texttt{has}(dtype)?\uparrow\langle NULL, aname\rangle.\\ \uparrow(content, dtype).dtype\downarrow\langle content\rangle.dtype\downarrow().\\ \uparrow\langle ACK\rangle.\mathbf{0}\end{array}\right) \tag{8}$$

The InfoStation will act as follows. First, it receives a request from $Lectreq_i$, then it checks the availability of the lecture in its cache by sending a request to the $Cache_i$ ambient, i.e.

$$\begin{array}{l}!Lectreq_i\downarrow(lectid, uid, dtype, aname).\\ Cache_i\downarrow\langle lectid, uid, dtype, aname\rangle.\mathbf{0}\end{array} \tag{9}$$

If the cache replies with the content of the lecture, it will send a request to the InfoStation Centre with a flag set to 1 (meaning that the requested lecture exists in its cache) asking whether the user is currently taking a mTest. If the user is taking a mTest, then the mLecture service request must be denied. If the cache did reply with NULL as lecture's content, then the infostation will send a request to the InfoStation Centre with the flag set to 0 (meaning that the lecture does not exist in its cache) asking for both the requested lecture and to check whether the user is taking a mTest. This behaviour of the IS is modelled as:

$$!Cache_i\downarrow(lectid, uid, dtype, creply, aname).\left(\begin{array}{l}\neg(creply = NULL)?\ ISC :: \langle lectid, uid, dtype,\\ aname, 1\rangle.\mathbf{C}\mid\\ \\ (creply = NULL)?\ ISC :: \langle lectid, uid, dtype,\\ aname, 0\rangle.\mathbf{N}\end{array}\right) \tag{10}$$

where $\mathbf{C}$ and $\mathbf{N}$ are defined as follows:

$$\mathbf{C} \;\widehat{=}\; ISC :: (lectid, reply, aname).\left(\begin{array}{l}(reply = OK \wedge \texttt{has}(aname))?\\ Lectreq_i\downarrow\langle lectid, creply, aname\rangle.\mathbf{0}\mid\\ \\ (reply = OK \wedge \neg\texttt{has}(aname))?\\ ISC :: \langle lectid, uid, dtype, aname, 0\rangle.\mathbf{0}\mid\\ \\ (reply = DENIED \wedge \texttt{has}(aname))?\\ Lectreq_i\downarrow\langle lectid, reply, aname\rangle.\mathbf{0}\end{array}\right)$$

and

$$\mathbf{N} \;\widehat{=}\; ISC :: (lectid, aname, reply).\left(\begin{array}{l}(\neg(reply = DENIED) \wedge \texttt{has}(aname))?\\ Cache_i\downarrow\langle lectid, reply, dtype\rangle.Lectreq_i\downarrow\langle lectid,\\ reply, aname\rangle.\ Cache_i\downarrow(ack).\mathbf{0}\mid\\ \\ (\neg(reply = DENIED) \wedge \neg\texttt{has}(aname))?\\ Cache_i\downarrow\langle lectid, reply, dtype\rangle.ISC :: \langle lectid, uid,\\ dtype, aname, 0\rangle.\ Cache_i\downarrow(ack).\mathbf{0}\mid\\ \\ (reply = DENIED \wedge \texttt{has}(aname))?\\ Lectreq_i\downarrow\langle lectid, reply, aname\rangle.\mathbf{0}\end{array}\right)$$

Thus, the whole behaviour of an infostation IS_i is modelled as

$$P_{IS_i} \;\widehat{=}\; \text{Eq. (3)} \mid \text{Eq. (9)} \mid \text{Eq. (10)}. \tag{11}$$

4.2.2 *InfoStation Centre*

A model of the ISC is depicted in Figure 5. It encompasses ambients modelling users' accounts and named after the users' IDs (e.g. 301 and 302 in Figure 5); an ambient named *Lectures* that contains all the lecture ambients, each named after the corresponding lecture ID (e.g. Lect001 and Lect002 in Figure 5). Each lecture ambient contains three persistent memory cells named $Phone$, PDA and PC; each storing the lecture's version suitable for the corresponding type of device. The mTest service is not represented as we are only dealing with the mLecture service in this paper. These components of the ISC are formalised below.

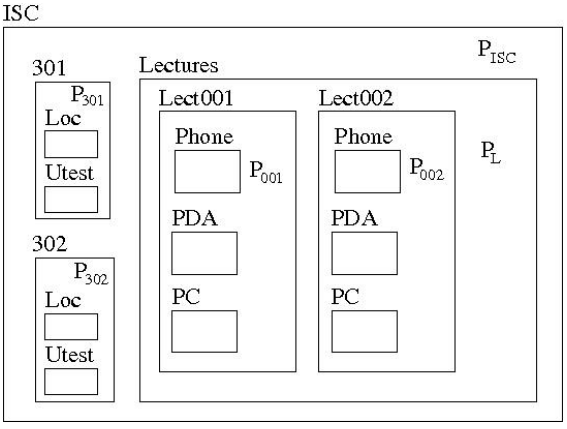


Figure 5: A model of Infostation Centre

Users' accounts. An ambient modelling a user's account contains two ambients named Loc and $Utest$. Each of these two ambients models a persistent memory cell 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). We understand by location of a user the IS the user is registered with. The behaviour of the Loc ambient and the $Utest$ ambient are specified exactly as in Eq. (1), with appropriate initial values. The ISC requests the value of any of these cells by sending the name Loc or $Utest$ to the user's account ambient (see Eq. (15)) which then can *get* (i.e. read) the value of the corresponding child ambient as follows, where the parameter x is the corresponding child ambient name:

$$!ISC \uparrow(x).x\downarrow\langle\rangle.x\downarrow(y).\ ISC \uparrow\langle y\rangle.\mathbf{0} \tag{12}$$

The user's account ambient can also *put* (i.e. write) 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:

$$!ISC \uparrow(x,n).x\downarrow\langle n\rangle.x\downarrow().ISC \uparrow\langle x, ACK\rangle.\mathbf{0} \tag{13}$$

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

$$P_{uid} \ \widehat{=}\ \text{Eq. (12)} \mid \text{Eq. (13)}.$$

Lectures. As mentioned above, this ambient contains all the lectures that are available in the mLecture service. Each lecture has a unique ID and the corresponding ambient is named after that ID. The behaviour of a lecture ambient is specified in Eq. (8). The $Lectures$ ambient behaves exactly as the cache ambient specified in Eq. (7)

Infostation centre. We now formalise the behaviour of the ISC when it receives a request from an IS. We are interested in two types of request in this paper: an AAA request and a lecture request.

When the ISC receives an AAA request from an IS, it updates the user's account with its new location and then replies to that IS with an acknowledge along with a list of available services. For the sake of simplicity, the service list is represented by the name '$SLIST$'. This is modelled by the following process:

$$\begin{array}{l} !IS_i :: (aaareq, uid, dtype, aname).uid\downarrow\langle IS_i\rangle.uid\downarrow(ack). \\ IS_i :: \langle ack, aname, SLIST\rangle.\mathbf{0} \end{array} \tag{14}$$

After receiving a lecture request, the ISC checks whether the user requesting the service is currently taking a mTest. This is done by it sending a message to the corresponding user's account ambient. That user's account ambient reply with 0 for 'No' and 1 for 'Yes'. If the reply is 'Yes' then the ISC fetches the current location of the user and forward a 'DENIED' message to that location. If the reply is 'No' and the flag is set to 1, a 'OK' message is forwarded to the current user location; otherwise (i.e. reply is 'No' and the flag is set to 0), the ISC fetches the appropriate lecture version for the user device and sends it to the current user location. This behaviour is represented by the following process:

$$\begin{array}{l} !IS_i :: (lectid, uid, dtype, aname, flag).uid\downarrow\langle Utest\rangle. \\ uid\downarrow(y).\ (P_{ISC1} \mid P_{ISC2}) \end{array} \tag{15}$$

where

$$\begin{array}{l} P_{ISC1} \ \widehat{=}\ (y=1)?uid\downarrow\langle Loc\rangle.uid\downarrow(z). \\ z :: \langle lectid, DENIED, aname\rangle.\mathbf{0} \end{array}$$

and

$$P_{ISC2} \ \widehat{=}\ \left(\begin{array}{l} (y=0 \wedge flag=1)?uid\downarrow\langle Loc\rangle.uid\downarrow(z). \\ z :: \langle lectid, OK, aname\rangle.\mathbf{0} \mid \\ (y=0 \wedge flag=0)?Lectures\downarrow\langle lectid, uid, \\ dtype, aname\rangle.Lectures\downarrow(lectid, uid, \\ dtype, reply, aname).uid\downarrow\langle Loc\rangle.uid\downarrow(z). \\ z :: \langle lectid, reply, aname\rangle.\mathbf{0} \end{array}\right)$$

So the whole behaviour of the ISC is modelled by the following process:

$$P_{ISC} \ \widehat{=}\ \text{Eq. (14)} \mid \text{Eq. (15)}.$$

5. VALIDATION

Now that a formal model of the infostation-based mLearning system has been presented, we show how this model can be used to validate the properties of the mLearning system. Lamport proposed two main classes of system's properties: *safety properties*, which state that 'nothing bad will happen'; and *liveness properties*, which assert that 'something good will happen, eventually'. In the light of this classification, we wish to establish the liveness property that every lecture request from a user is eventually replied to by the system, provided that the user does not become infinitely unavailable after that request has been made.

THEOREM 5.1. *Every user's lecture request will eventually get a reply, provided that the user stays long enough in the system.*

PROOF. The proof of this theorem is based on the reduction semantics of `CCA` given by a congruence relation '$\equiv$' defined in Table 9 and a reduction relation '$\rightarrow$' defined in Table 10, both in [13].

In this proof we will assume, without loss of generality, that the user is using a laptop (PC) to access the system from an infostation IS_i. Given that the user is mobile, the following cases must be considered:

1. the user sends the request and waits for the reply in the same infostation IS_i (i.e. the user may move around within the range of the infostation)

2. the user sends the request and move into a different infostation IS_j, $j \neq i$.

In Case 1, the user behaviours can be modelled by the following ambient:

$$PC303[Lectreq_i :: \langle Lect001, 303, PC, PC303\rangle. \\ Lectreq_i :: (lectid, reply).\mathbf{0}] \quad (16)$$

This ambient sends a lecture request to the $Lectreq_i$ ambient and waits for a reply from that ambient, and then terminates. The lecture request contains the following information: (i) the lecture ID, $Lect001$; (ii) the user ID, 303; (iii) the device type, PC; and (iv) the name of the ambient to reply to, $PC303$.

The behaviours of the $Lectreq_i$ ambient is modelled by the process P_{L_i} in Eq. (6), which basically receives a lecture request from a sibling ambient (e.g. a user device), forward the request to the infostation IS_i, get the reply from that infostation and forwards it to the very ambient which initiated the request. How the infostation IS_i interacts with the $Lectreq_i$ ambient is specified in Eq. (9). These interactions between a user device, the $Lectreq_i$ ambient and the infostation IS_i can be expressed as a sequence of derivations using the reduction relation $\rightarrow$. Because of the space limit, we cannot give the full sequence of derivations in this paper. For illustration, the following sequence of derivations describes how a lecture request sent by the user gets to the infostation to be processed:

Eq. (9) | Eq. (16) | $Lectreq_i[P_{L_i}]$

$\rightarrow$ {Rule (Red Com S2) in Table 10; the request is sent to the $Lectreq_i$ ambient}

$$\text{Eq. (9)} \mid PC303[Lectreq_i :: (lectid, reply).\mathbf{0}] \mid \\ Lectreq_i \left[\begin{array}{l} IS_i \uparrow \langle Lect001, 303, PC, PC303\rangle.\mathbf{0} \mid \\ \\ !:: (lectid, uid, dtype, aname). \\ IS_i \uparrow \langle lectid, uid, dtype, aname\rangle.\mathbf{0} \mid \\ \\ !IS_i \uparrow (lectid, reply, aname). \\ aname :: \langle lectid, reply\rangle.\mathbf{0} \end{array}\right]$$

$\rightarrow$ {Rule (Red Com R6) in Table 10; the $Lectreq_i$ ambient forwards the request to the infostation}

$$\text{Eq. (9)} \mid Cache_i \downarrow \langle Lect001, 303, PC, PC303\rangle.\mathbf{0} \\ \mid PC303[Lectreq_i :: (lectid, reply).\mathbf{0}] \mid \\ Lectreq_i \left[\begin{array}{l} !:: (lectid, uid, dtype, aname). \\ IS_i \uparrow \langle lectid, uid, dtype, aname\rangle.\mathbf{0} \mid \\ \\ !IS_i \uparrow (lectid, reply, aname). \\ aname :: \langle lectid, reply\rangle.\mathbf{0} \end{array}\right]$$

At this stage, the infostation IS_i has received a lecture request from the $Lectreq_i$ ambient and is willing to check with the $Cache_i$ ambient whether it has the requested lecture for the specified type of device. The behaviour of the $Cache_i$ ambient is specified by the process P_{c_i} in Eq. (7). If the requested lecture $Lect001$ exists in the cache for the specified type of device, then the $Cache_i$ ambient gets a copy of the lecture for the specified type of device by interacting with the child ambient named $Lect001$ whose behaviour is specified by the process $P_{Lect001}$ as in Eq. (8). It can also be seen from Eq. (7) and Eq. (8) that if the requested lecture $Lect001$ does not exist in the cache for the specified type of device, then a reply message '$NULL$' is forwarded to the infostation IS_i. So in either situation, the infostation IS_i receives a reply from the cache.

Once a reply is received from the $Cache_i$ ambient, the infostation IS_i contacts the infostation centre ISC as specified by the process in Eq. (10). How the ISC reacts is modelled by Eq. (15); it replies with a '$DENIED$' message if the user requesting the lecture is currently using a mTest service, otherwise it replies with a 'OK' message and possibly a copy of the requested lecture if it is not available locally in IS_i's cache. How each of these types of reply is handled by the IS_i is modelled by the component C and N in Eq. (10). One can see that for every case where the user is still in the range of the infostation IS_i (i.e. the context expression '$\texttt{has}(aname)$' holds), the infostation IS_i sends a reply to the $Lectreq_i$ ambient which subsequently forwards the reply to the user device as specified in Eq. (5). This completes the proof of Case 1.

The proof of Case 2 can be done in a similar manner as in Case 1, with the user behaviours specified as in Eq. (17), where $i \neq j$, i.e. the processes of sending the request and receiving the reply are done in different infostations.

$$PC303 \left[\begin{array}{l} Lectreq_i :: \langle Lect001, 303, PC, PC303\rangle.\texttt{out}. \\ \texttt{in}\ IS_j.AAAreq_j :: \langle 303, PC, PC303\rangle.\mathbf{0} \mid \\ \\ \texttt{at}(IS_j)?AAAreq_j :: (ack, slist). \\ Lectreq_j :: (lectid, reply).\mathbf{0} \end{array}\right] \quad (17)$$

This ambient sends a lecture request from the infostation IS_i, moves to a different infostation IS_j, registers with this infostation by sending an AAA request then waits for the acknowledgement of its registration. Once its registration has been confirmed, it then prompts to receive the reply to the lecture request and then terminates.

□

6. RELATED WORK

Milner [10] introduced bigraphical reactive systems (BRSs) as a unifying framework for designing models of concurrent and mobile systems. In [2], Birkedal et al. attempt the modelling of contextaware systems using bigraphical reactive systems (BRSs). They reach the conclusion that BRSs are not suitable for directly modelling context queries and propose plato-graphical models as an alternative. While in plato-graphical models contexts are queried through a proxy (also modelled as a BRS), in CCA primitives such as context-guarded prefix and process call are used to acquire context information. CONAWA [7] has been proposed as a calculus for context awareness. Like CCA, CONAWA is inspired by the π calculus. The syntax of CONAWA focuses on constructs which makes it possible to describe and navigate through context.

According to [12], formalising a context domain not only requires the vocabularies of concepts but relationships among them as well. This is the focus of ontology-based approaches [1, 12]. A UML specification of an infostation-based mLearning system was proposed in [5]. Although UML provides a number of advantages such as code generation and other system analyses using assistant tools, the lack of support for formal reasoning is one of its limitations for the design of critical systems.

7. CONCLUSION AND FUTURE WORK

This paper presented the formalisation of the policies of an Infostation-based mLearning system, with emphasis on the AAA service and mLecture service. This case study is yet another demonstration of

the expressiveness and the pragmatics of CCA which has been introduced recently as a suitable process calculus for modelling mobile and context-aware systems. It was also shown how properties of the mLearning system can be validated using the reduction semantics of CCA.

In future work, we will extend the formalisation of the mLearning system to encompass other mServices such as mTest and mTutorial. As customary in process calculi, we will also investigate type systems for verifying key properties of a mLearning system such as safety, security and privacy. It is also being developed a software tool for the execution and animation of CCA specifications.

8. REFERENCES

[1] W3C Web Ontology Working Group: The Web Ontology Language: OWL. http://www.w3.org/2001/sw/WebOnt/.

[2] L. Birkedal, S. Debois, E. Elsborg, T. Hildebr, and H. Niss. Bigraphical models of context-aware systems. In *IT University of Copenhagen*, pages 187–201. Springer-Verlag, 2006.

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

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

[5] 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.

[6] C. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[7] M. B. KjÃęrgaard and J. Bunde-Pedersen. Towards a formal model of context awareness. In *1st International Workshop on Combining Theory and Systems Building in Pervasive Computing, CTSB 2006*, 2006.

[8] R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.

[9] R. Milner. *Communication and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[10] R. Milner. Pure Bigraphs: structure and dynamics. *Information and Computation*, 204:60–122, 2006.

[11] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[12] A. Shehzad, H. Q. Ngo, K. A. Pham, and S. Y. Lee. Formal modeling in context aware systems. In *1st International Workshop on Modeling and Retrieval of Context (MRC 2004), Ulm, Germany*, 2004.

[13] F. Siewe, H. Zedan, and A. Cau. The Calculus of Context-aware Ambients. *Journal of Computer and System Science*, doi:10.1016/j.jcss.2010.02.003, 2010.