Journal of Computer and System Sciences 77 (2011) 597–620

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

The Calculus of Context-aware Ambients

François Siewe *, Hussein Zedan, Antonio Cau

Software Technology Research Laboratory, De Montfort University, The Gateway, Leicester, LE1 9BH, United Kingdom

ARTICLE INFO

Article history:
Received 15 September 2009
Received in revised form 17 December 2009
Available online 25 February 2010

Keywords:
Context-awareness
Process calculus
Mobility
Ambient
Pervasive
Ubiquitous
Computing
Systems

ABSTRACT

We present the Calculus of Context-aware Ambients (CCA in short) for the modelling and verification of mobile systems that are context-aware. This process calculus is built upon the calculus of mobile ambients and introduces new constructs to enable ambients and processes to be aware of the environment in which they are being executed. This results in a powerful calculus where both mobility and context-awareness are first-class citizens. We present the syntax and a formal semantics of the calculus. We propose a new theory of equivalence of processes which allows the identification of systems that have the same context-aware behaviours. We prove that CCA encodes the π-calculus which is known to be a universal model of computation. Finally, we illustrate the pragmatics of the calculus through many examples and a real-world case study of a context-aware hospital bed.

1. Introduction

Pervasive computing [1] is a new paradigm for next-generation distributed systems where computers disappear in the background of the users' everyday activities. In such a paradigm computation is performed on a multitude of small devices interconnected through a wireless network. Fundamental to pervasive computing is that any component (including user, hardware and software) can be *mobile* and that computations are *context-aware*. As a result, mobility and context-awareness are important features of any design framework for pervasive computing applications. Context-awareness requires applications to be able to sense aspects of the environment and use this information to adapt their behaviour in response to changing situations. These aspects of the environment that can influence the behaviour of an application constitute the *context* of that application. For example, *location-aware* applications use location information – such as where they are, what is nearby – to adapt themselves when the situation changes. Although location is an important aspect of context, context is more than location. Dey [2] defined context as:

> any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

It follows from this definition that the user, the user's location, the nearby people, available resources, the user activities, the user social situations, the user preferences, and device profiles to name a few are important aspects of the context of an application. Despite the advances in mobile computing, there is a fundamental lack of linguistic support and mechanisms for modelling context-awareness in mobile applications.

* Corresponding author.
E-mail addresses: fsiewe@dmu.ac.uk (F. Siewe), hzedan@dmu.ac.uk (H. Zedan), acau@dmu.ac.uk (A. Cau).

0022-0000/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2010.02.003

In order to address these issues, this paper proposes the Calculus of Context-aware Ambients (CCA in short), that is built upon the calculus of Mobile Ambients (MA in short) [3] and introduces new constructs to enable ambients and processes to be aware of the environment in which they are being executed. This results in a powerful calculus where mobility and context-awareness are first-class citizens. Our contributions are summarised as follows:

- We define a formal model of context (Section 4) for representing the contexts of CCA processes based on the hierarchical structure of ambients.
- We propose a logical language for expressing properties of the contexts of CCA processes (Section 5). We call a formula in this logic a *context expression*. Context expressions are used in CCA to constrain the environment in which a process is execution. This is done by guarding each individual action in a process by a context expression that must hold for the action to take place. In CCA a primitive action is also known as a *capability*. So, a *context-guarded capability* has the form $\kappa?M$ where the guard κ is a context expression and M is a capability; the capability M is performed only when the environment satisfies its guard κ. We give the semantics of context expressions in terms of a satisfaction relation with respect to our formal model of contexts.
- We give the syntax (Section 3) and formal semantics (Section 6) of CCA which supports context-aware mobile processes. It inherits the mobility model of MA. Its innovative features are (i) *context-guarded capability* as explained above; (ii) *process abstraction* as a mechanism for context provision; and (iii) *process call* as a mechanism for context acquisition.
- We propose a new contextual equivalence of processes (Section 8) which is based on Morris' theory of equivalence [4]. The innovative feature of our approach resides in our notion of elementary observation which is defined as the occurrence in a process of a top-level ambient whose name is not restricted and which is aware of the contexts described by a context expression. Two processes are then equivalent if they admit the same elementary observations whenever they are inserted inside any arbitrary enclosing process, i.e. the two processes have the same *context-aware behaviours*. We prove that our contextual equivalence is a congruence.
- We prove that CCA encodes the π-calculus (Section 9) which is known to be a universal model of computation [5], and illustrate the pragmatics of the calculus through small examples (Section 2) and a real-world case study of a context-aware hospital bed (Section 7).

2. Motivation and examples

Context-awareness requires applications to be able to adapt themselves to the environment in which they are being used such as user, location, nearby people and devices, and user's social situations. In this section we use small examples to illustrate the ability of CCA to model applications that are context-aware.

2.1. Smart phone example

Consider a smart phone which automatically switches modes depending on the context of use. For example, the phone is silent (i.e. does not ring but only vibrates on incoming calls or text messages) when the user is at a conference or in a meeting; or diverts calls to a voice mail system if the user is with friends. We model incrementally such a smart phone system using the notion of ambient.

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. Syntactically, an ambient is represented by a structure of the form:

$$n[P_1 \mid P_2 \mid \cdots \mid P_k],$$

where n is the name of the ambient and each P_i, $1 \leqslant i \leqslant k$, is a process or a *child* ambient. The symbol '|' denotes the parallel composition of processes. The pair of square brackets delimits the boundary of the ambient. We call this representation the *textual* representation of an ambient. The *graphical* representation of an ambient is a box labelled with the ambient name n and containing the processes P_i, $1 \leqslant i \leqslant k$, as follows:

$$\overset{n}{\boxed{P_1 \mid P_2 \mid \cdots \mid P_k}}$$

The graphical representation is used as visual aids to increase readability and understanding.

We model the smart phone device as an ambient named *phone*:

$$\overset{phone}{\boxed{P_{phone}}}$$

where P_{phone} is a process specifying the functionality of the phone. The user, Bob say, carrying the phone can be modelled as a parent ambient named *bob*:

$$\overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid P_{bob}}}$$

where P_{bob} is a process describing the behaviour of the user Bob and other devices that he is carrying with him.

The location of the phone's user can also be modelled as a parent ambient. For example the following ambient named *conf* models the location of the phone's user Bob when he is in a conference room:

$$\overset{conf}{\boxed{\overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid P_{bob}}} \mid P_{conf}}}$$

where the process P_{conf} models the remaining part of the conference room.

Suppose the conference room is next to the reception room which is represented by an ambient named *recept* say; this is modelled by composing in parallel the corresponding ambients, i.e.

$$\overset{conf}{\boxed{P_{conf}}} \mid \overset{recept}{\boxed{P_{recept}}}$$

In CCA there are two mobility capabilities (or primitives) that enable ambients to move autonomously from one location to another: these are the capability `in` and the capability `out`. The former takes a single argument which is the name of the destination ambient and enables an ambient to move into the destination ambient. However, this capability is performed only if the ambient which is willing to move is next to the destination ambient. The latter takes no arguments and enables an ambient to move out of its parent ambient; the capability is performed only if the parent ambient exists. As customary, the semantics of a process is defined using the reduction relation '$\rightarrow$' which states how the process is executed.

For example, the user Bob can move out of the conference room by performing the capability `out` as stated by the following reduction, where Q is a continuation process:

$$\overset{conf}{\boxed{\overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid \mathtt{out}.\mathtt{in}\ recept.Q}} \mid P_{conf}}} \mid \overset{recept}{\boxed{P_{recept}}} \quad\rightarrow\quad \overset{conf}{\boxed{P_{conf}}} \mid \overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid \mathtt{in}\ recept.Q}} \mid \overset{recept}{\boxed{P_{recept}}}$$

Then Bob can move in the reception room by performing the capability `in` as illustrated by the following reduction:

$$\overset{conf}{\boxed{P_{conf}}} \mid \overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid \mathtt{in}\ recept.Q}} \mid \overset{recept}{\boxed{P_{recept}}} \quad\rightarrow\quad \overset{conf}{\boxed{P_{conf}}} \mid \overset{recept}{\boxed{\overset{bob}{\boxed{\overset{phone}{\boxed{P_{phone}}} \mid Q}} \mid P_{recept}}}$$

We just showed how devices, users, user's locations and mobility can be modelled in CCA, using the notion of ambient inherited from MA [3]. We now consider some context-awareness scenarios of the smart phone application and show how these scenarios are expressed in CCA. It is assumed that the phone can be configured to function in the following modes: *normal*, to vibrate and ring when there is an incoming call; *silent*, to vibrate but not to ring when there is an incoming call; *divert*, to divert to a voice mail system all incoming calls; *reject*, to reject all incoming calls.

Scenario 1. *The phone must divert all incoming calls when the user is with a friend, Alice.*

To implement this scenario, the phone needs to be able to detect the people with whom its user is; then switch into the appropriate mode if Alice is among them. We say that an ambient is with another ambient if the two ambients are located at the same location, i.e. are sibling in ambient hierarchy. We let $\texttt{user\_with}(n)$ denote a context expression (i.e. a predicate over contexts) which is true if the parent of the ambient evaluating that context expression is a sibling of the ambient n. Its formal definition is given by Eq. (8) on p. 607. We assume that the parent ambient of the phone is the user. We suppose that the phone switches to the mode x by performing a capability $\texttt{switchto}\langle x\rangle$ whose details are not given here to focus on the context-awareness aspect of the application. However, the formal definition of the process abstraction `switchto` is given in Eq. (18)-*d* on p. 616.

We used the notion of *context-guarded capability* to model context-awareness. A *context-guarded capability* has the form $\kappa?M$ where κ is a context expression and M is an action capability (e.g. `in` or `out`); the capability M is performed only if the current context satisfies the guard κ. A *context-guarded prefix* is a process of the form $\kappa?M.P$, where P is a continuation process; when the context of this process satisfies the context expression κ, it performs the capability M and continues like P. We now specify Scenario 1 as follows:

$$\mathit{phone}\,\boxed{!\,\texttt{user\_with}(\mathit{alice})?\texttt{switchto}\langle \mathit{divert}\rangle.\mathbf{0}} \tag{1}$$

where the symbol '!' is the replication operator as defined in the π-calculus [6] and *alice* is the name of the ambient modelling Alice.

Indeed, a replication $!P$ denotes a process which can always create a new copy of P, i.e. the process $!P$ is equivalent to the process $P \mid !P$. Replication is a convenient way of representing iteration and recursion. The process $\mathbf{0}$ is the nil process; it does nothing and terminates immediately. The sequential composition of processes is denoted by the symbol '.' (i.e. dot). In Eq. (1), the ambient *phone* repeatedly senses the context of its user to check whether its user is co-located with the ambient *alice*; if so the phone switches into the *divert* mode where all incoming calls are diverted to a voice mail system.

Scenario 2. *The phone must vibrate but not ring to alert for incoming calls when the user is at the conference room.*

Let $\texttt{user\_at}(n)$ denote a context expression which is true if the parent of the ambient evaluating the expression is located in the ambient n. Its formal meaning is defined in Eq. (7) on p. 607. We assume that the parent ambient of the phone is the user. Scenario 2 can be specified as follows:

$$\mathit{phone}\,\boxed{!\,\texttt{user\_at}(\mathit{conf})?\texttt{switchto}\langle \mathit{silent}\rangle.\mathbf{0}} \tag{2}$$

This means that if the phone's user is in the conference room the phone vibrates on incoming calls but does not ring.

If we combine Scenario 1 and Scenario 2, the specification of the phone becomes:

$$\mathit{phone}\,\boxed{\begin{array}{ll} !\,\texttt{user\_with}(\mathit{alice})?\texttt{switchto}\langle \mathit{divert}\rangle.\mathbf{0} & \mid \\ !\,(\texttt{user\_at}(\mathit{conf}) \wedge \neg\texttt{user\_with}(\mathit{alice}))?\texttt{switchto}\langle \mathit{silent}\rangle.\mathbf{0} & \end{array}}$$

This means that if the phone's user is in the conference room and its friend Alice is not there then the phone vibrates on incoming calls but does not ring. However, calls will be diverted if both the phone's user and Alice are in the conference room.

Of course the phone should be able to ring and receive calls when the user is neither in the conference room, nor with Alice. This is specified as follows:

$$\mathit{phone}\,\boxed{\begin{array}{ll} !\,\texttt{user\_with}(\mathit{alice})?\texttt{switchto}\langle \mathit{divert}\rangle.\mathbf{0} & \mid \\ !\,(\texttt{user\_at}(\mathit{conf}) \wedge \neg\texttt{user\_with}(\mathit{alice}))?\texttt{switchto}\langle \mathit{silent}\rangle.\mathbf{0} & \mid \\ !\,(\neg\texttt{user\_at}(\mathit{conf}) \wedge \neg\texttt{user\_with}(\mathit{alice}))?\texttt{switchto}\langle \mathit{normal}\rangle.\mathbf{0} & \end{array}} \tag{3}$$

The next section introduces additional constructs for modelling context-awareness in CCA.

2.2. Context-aware file editing agent

This example illustrates the use of *process abstraction* and *process call* as a mechanism for context provision and context acquisition, respectively. A process abstraction can be thought of as the declaration of a procedure in procedural programming languages and a process call as the invocation of a procedure.

A process abstraction links a name x to a process P using the following syntax:

$$x \rhd (\tilde{y}).P,$$

where $\tilde{y}$ is the list of formal parameters. A process call to this process abstraction has the following syntax:

$$x\langle\tilde{z}\rangle,$$

where $\tilde{z}$ is the list of the actual parameters. This process call behaves exactly like the process P where each actual parameter in $\tilde{z}$ is substituted for each occurrence of the corresponding formal parameter in $\tilde{y}$. In the smart phone example presented above, `switchto` is a process abstraction defined as in Eq. (18) on p. 616.

Suppose a software agent *agt* (here modelled as an ambient) is willing to edit a text file *foo*. This is done by calling a process abstraction named `edit` say, as follows:

$$\overset{\mathit{agt}}{\boxed{\uparrow\mathtt{edit}\langle \mathit{foo}\rangle.\mathbf{0}}}$$

where the symbol '↑' indicates that the `edit` process called here is the one that is defined in the parent ambient of the calling ambient *agt*. Now suppose agent *agt* has migrated to a computing device *win* running Microsoft Windows operating system:

$$\overset{\mathit{win}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{notepad}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\uparrow\mathtt{edit}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}}$$

On this machine, the process abstraction `edit` is defined to launch the text editor `notepad` as follows:

$$\mathtt{edit}\rhd(y).\mathtt{notepad}\langle y\rangle.\mathbf{0}.$$

So the request of the agent *agt* to edit the file *foo* on this machine will open that file in notepad according to the following reduction:

$$\overset{\mathit{win}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{notepad}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\uparrow\mathtt{edit}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}} \quad\rightarrow\quad \overset{\mathit{win}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{notepad}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\mathtt{notepad}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}}$$

Note that the command `notepad` has replaced the command `edit` in the calling ambient *agt*.

Now assume the agent *agt* first moved to a computer *lin* running Linux operating system:

$$\overset{\mathit{lin}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{emacs}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\uparrow\mathtt{edit}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}}$$

On this computer, the command `edit` is configured to launch `emacs`. So in this context, the file *foo* will be opened in emacs as illustrated by the following reduction:

$$\overset{\mathit{lin}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{emacs}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\uparrow\mathtt{edit}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}} \quad\rightarrow\quad \overset{\mathit{lin}}{\boxed{\begin{array}{l}\mathtt{edit}\rhd(y).\mathtt{emacs}\langle y\rangle.\mathbf{0}\mid\\ \qquad\overset{\mathit{agt}}{\boxed{\mathtt{emacs}\langle \mathit{foo}\rangle.\mathbf{0}}}\end{array}}}$$

Our agent *agt* might have even moved to a site where the command `edit` is not *available* because no process abstraction of that name is defined. In this case the agent *agt* will not be able to edit the file *foo* at this site and might consider moving to a nearby computer to do so.

The next section details the syntax and informal semantics of `CCA` processes.

Table 1
Syntax of CCA processes and capabilities.

P, Q, R ::=		**Process**	α ::=		**Locations**
	$\mathbf{0}$	inactivity		$\uparrow$	any parent
	$P \mid Q$	parallel composition		$n\uparrow$	parent n
	$(\nu n)\ P$	name restriction		$\downarrow$	any child
	$n[P]$	ambient		$n\downarrow$	child n
	$!P$	replication		$::$	any sibling
	$\kappa?M.P$	context-guarded action		$n::$	sibling n
	$x \rhd (\tilde{y}).P$	process abstraction		ϵ	locally

M ::=		**Capabilities**
	del n	delete n
	in n	move in n
	out	move out
	$\alpha\ x\langle\tilde{y}\rangle$	process call
	$\alpha\ (\tilde{y})$	input
	$\alpha\ \langle\tilde{y}\rangle$	output

3. Syntax of processes and capabilities

This section introduces the syntax of the language of CCA. Like in the π-calculus [6,7], the simplest entities of the calculus are *names*. These are used to name for example ambients, locations, resources and sensors data. 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. We distinguish three main syntactic categories: processes P, capabilities M and context expressions κ.

The syntax of processes and capabilities is given in Table 1 where P, Q and R stand for processes, and M for capabilities. The first five process primitives (inactivity, parallel composition, name restriction, ambient and replication) are inherited from MA [3]. The process $\mathbf{0}$ does nothing and terminates immediately. The process $P \mid Q$ 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. Replication was first introduced by Milner in the π-calculus [6]. 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

$$\overset{n}{\boxed{P}}$$

The graphical representation highlights the nested structure of ambients.

CCA departs from MA and other processes calculi such as [8–10] with the notion of *context-guarded capabilities*, whereby a capability is guarded by a context expression which specifies the condition that must be met by the environment of the executing process. A process prefixed with a context-guarded capability is called a *context-guarded prefix* and it has the form $\kappa?M.P$. Such a process waits until the environment satisfies the context expression κ, then performs the capability M and continues like the process P. The process learns about its context (i.e. its environment) by evaluating the guard. The use of context-guarded capabilities 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). The syntax and the semantics of context expressions are given in Section 5. We let $M.P$ denote the process $\mathbf{True}?M.P$, where **True** is a context expression satisfied by all context.

A process abstraction $x \rhd (\tilde{y}).P$ 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*. There must be as many actual parameters as there are formal parameters to the process abstraction being called. 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. 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.

Example 3.1.

i) The capability $\uparrow x\langle\tilde{z}\rangle$ calls the process abstraction x defined in the parent of the calling ambient;
ii) The capability $::x\langle\tilde{z}\rangle$ calls the process abstraction x defined in one of the siblings of the calling ambient;

iii) The capability $n\downarrow x\langle\tilde{z}\rangle$ calls the process abstraction x defined in a child ambient named n of the calling ambient;
iv) The capability $x\langle\tilde{z}\rangle$ calls the process abstraction x defined in the calling ambient.

In CCA, an ambient provides context by (re)defining process abstractions to account for its specific functionality. Ambients can interact with each other by making process calls. Because ambients are mobile, the same process call, e.g. $\uparrow x\langle\tilde{z}\rangle$, may lead to different behaviours depending on the location of the calling ambient. So process abstraction is used as a mechanism for context provision while process call is a mechanism for context acquisition.

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. Similarly to a process call, an ambient can send message to any parent, i.e. $\uparrow\langle\tilde{z}\rangle$; a specific parent n, i.e. $n\uparrow\langle\tilde{z}\rangle$; any child, i.e. $\downarrow\langle\tilde{z}\rangle$; a specific child n, i.e. $n\downarrow\langle\tilde{z}\rangle$; any sibling, i.e. $::\langle\tilde{z}\rangle$; a specific sibling n, i.e. $n::\langle\tilde{z}\rangle$; or itself, i.e. $\langle\tilde{z}\rangle$.

An *input prefix* is a process of the form $\alpha\ (\tilde{y}).P$, where $\tilde{y}$ is a list of variable symbols and P is a continuation process. It receives a list of names $\tilde{z}$ from the location α and continues like the process $P\{\tilde{y}\leftarrow\tilde{z}\}$, where $P\{\tilde{y}\leftarrow\tilde{z}\}$ is the process P in which each name in the list $\tilde{z}$ is substituted for each occurrence of the corresponding variable symbol in the list $\tilde{y}$.

The mobility capabilities in and out are defined as in MA [3] with the exception that the capability out has no explicit parameter in CCA, the implicit parameter being the current parent (if any) of the ambient performing the action. An ambient that performs the capability in n moves into the sibling ambient n. The capability out moves the ambient that performs it out of that ambient parent. Obviously, a root ambient, i.e. an ambient with no parents, cannot perform the capability out. The capability del n deletes an ambient of the form $n[\mathbf{0}]$ situated at the same level as that capability, i.e. the process 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. It is a constrained version of the capability open used in MA to unleash the content of an ambient. As mentioned in [10], the open capability brings about serious security concerns in distributed applications, e.g. it might open an ambient that contains a malicious code. Unlike the capability open, the capability del is secure because it only opens ambients that are empty, so no risk of opening a virus or a malicious ambient.

4. Context model

In CCA the notion of *ambient*, inherited from MA, is the basic structure used to model entities of a context-aware system such as: a user, a location, a computing device, a software agent or a sensor. As described in Table 1, an ambient has a name, a boundary, a collection of local processes and can contain other ambients. Meanwhile, an ambient can move from one location to another by performing the mobility capabilities in and out. So the structure of a CCA process, at any time, is a hierarchy of nested ambients. This hierarchical structure changes as the process executes. In such a structure, the context of a sub-process is obtained by replacing in the structure that sub-process by a placeholder '$\odot$'. 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 m is $P \mid n[Q \mid \odot]$. Following are examples of contexts in the smart phone system described in Section 2.

Example 4.1. The following context is the context of the smart phone carried by Bob when Bob is inside the conference room with Alice:

$$e_1 \mathrel{\widehat{=}} conf\big[P \mid bob[\odot] \mid alice[Q]\big],$$

where P models the remaining part of the internal context of the conference room and Q the internal context of the ambient *alice*. We assume that there is only one ambient named *alice* in the conference room.

Example 4.2. If Bob is inside the conference room while Alice is outside that room, the context of the smart phone carried by Bob can be described as follows:

$$e_2 \mathrel{\widehat{=}} alice[Q] \mid conf\big[P \mid bob[\odot]\big].$$

Example 4.3. Bob might carry with him another device, a PDA say, while inside the conference room. In this case the context of the smart phone can be modelled as:

$$e_3 \mathrel{\widehat{=}} conf\big[P' \mid bob\big[\odot \mid pda[R]\big]\big],$$

where P' models the remaining part of the internal context of the conference room, *pda* is the name of the ambient modelling the PDA device and R specifies the functionality of the PDA.

Our context model is depicted by the grammar in Table 2, where the symbol E stands for context (environment), n ranges over names and P ranges over processes (as defined in Table 1). The context $\mathbf{0}$ is the empty context, also called the *nil* context. It contains no context information. The position of a process in that process' context is denoted by the symbol $\odot$. This is a special context called the *hole context*. The context $(\nu n)\ E$ means that the scope of the name n is limited to the context E. The context $n[E]$ means that the internal environment of the ambient n is described by the context E. The context $E \mid P$ says that the process P runs in parallel with the context E, and so E is part of process P's context.

Table 2
Syntax of contexts.

E	::=		**Context**
		$\mathbf{0}$	nil
		$\odot$	hole
		$n[E]$	location
		$E \mid P$	parallel composition
		$(\nu n)\ E$	restriction

Table 3
Algebraic semantics of contexts.

$E \mid \mathbf{0} = E$	(Cont-0)
$E_1 \mid E_2 = E_2 \mid E_1$	(Cont-1)
$E_1 \mid (E_2 \mid E_3) = (E_1 \mid E_2) \mid E_3$	(Cont-2)
$(\nu n)\ (\nu m)\ E = (\nu m)\ (\nu n)\ E$	(Cont-3)
$(\nu n)\ E_1 \mid E_2 = (\nu n)\ (E_1 \mid E_2)$ if $n \notin \mathbf{fn}(E_2)$	(Cont-4)
$(\nu n)\ m[E] = m[(\nu n)\ E]$ if $n \neq m$	(Cont-5)
$(\nu n)\ \mathbf{0} = \mathbf{0}$	(Cont-6)
$E_1 = E_2 \quad \Rightarrow \quad (\nu n)\ E_1 = (\nu n)\ E_2$	(Cont-7)
$E_1 = E_2 \quad \Rightarrow \quad E_1 \mid E_3 = E_2 \mid E_3$	(Cont-8)
$E_1 = E_2 \quad \Rightarrow \quad n[E_1] = n[E_2]$	(Cont-9)

Definition 4.1 *(Ground context).* A ground context is a context containing no holes.

Note that a context contains zero or one hole; and that a ground context is a process. We do not allow multi-hole contexts because they are not suitable to our purpose.

Definition 4.2 *(Context evaluation).* Let E_1 and E_2 be contexts. The evaluation of the context E_1 at the context E_2, denoted by $E_1(E_2)$, is the context obtained by replacing the hole in E_1 (if any) by E_2, viz

$$E_1(E_2) = \begin{cases} E_1 & \textit{if } E_1 \textit{ is a ground context}, \\ E_1\{\odot \leftarrow E_2\} & \textit{otherwise}, \end{cases}$$

where $E_1\{\odot \leftarrow E_2\}$ is the substitution of E_2 for $\odot$ in E_1.

The hole $\odot$ plays an important role in our context model. In fact a context E containing a single hole represents the environment of a process P in the process $E(P)$.

Example 4.4. A process modelling Bob using a smart phone in the conference room with Alice can be specified as:

$$e_1\big(phone[S]\big) \mathrel{\widehat{=}} conf\big[P \mid bob\big[phone[S]\big] \mid alice[Q\,]\big],$$

where e_1 is the context specified in Example 4.1 and S is the specification of the smart phone (see Section 2.1).

Example 4.5. A process modelling Bob using a PDA in the conference room can be specified as:

$$e_3(\mathbf{0}) \mathrel{\widehat{=}} conf\big[P' \mid bob\big[pda[R]\big]\big],$$

where e_3 is the context specified in Example 4.3.

An algebraic semantics of contexts is given in Table 3 in terms of equalities, where the set $\mathbf{fn}(E)$ of free names in a context E is defined as for processes (see Table 11 in Appendix A). The first three equalities say that the parallel composition of contexts has a unit element $\mathbf{0}$, and is commutative and associative, respectively. The equalities (Cont-3) to (Cont-6) are related to the manipulation of scopes. The last set of algebraic rules state that equality propagates across scopes, parallel composition and ambient nesting, respectively.

In order to navigate through the hierarchical structure of context, we define a spatial reduction relation $\downharpoonleft\!\downharpoonright$ for context as follows:

$$E_1 \downharpoonleft\!\downharpoonright E_2 \quad \text{iff} \quad \text{exist a name } n \text{ and a context } E_3 \text{ such that } E_1 = \big(n[E_2] \mid E_3\big). \tag{4}$$

Eq. (4) says that the context E_1 contains the context E_2 within exactly one level of nesting. Theorem 4.1 asserts that the spatial reduction relation $\downharpoonleft\!\downharpoonright$ is closed under equality.

Theorem 4.1.

$$E_1 = E_2,\ E_2 \downharpoonleft\!\downharpoonright E_2',\ E_2' = E_1' \quad \Rightarrow \quad E_1 \downharpoonleft\!\downharpoonright E_1'.$$

Table 4
Syntax of context expressions.

κ	$::=$		**Context expressions**
		True	true
		$n = m$	name match
		$\bullet$	hole
		$\neg\kappa$	negation
		$\kappa_1 \mid \kappa_2$	parallel composition
		$\kappa_1 \wedge \kappa_2$	conjunction
		$n[\kappa]$	location
		$\mathtt{new}(n, \kappa)$	revelation
		$\oplus\kappa$	spatial next modality
		$✧\kappa$	somewhere modality
		$\exists x.\kappa$	existential quantification

Table 5
Satisfaction relation for context expressions.

E	$\models$	**True**			(Sat-true)
E	$\models$	$n = n$			(Sat-match)
E	$\models$	$\bullet$	iff	$E = \odot$	(Sat-hole)
E	$\models$	$\neg\kappa$	iff	$E \not\models \kappa$	(Sat-neg)
E	$\models$	$\kappa_1 \mid \kappa_2$	iff	exist E_1, E_2 such that $E = E_1 \mid E_2$ and $E_1 \models \kappa_1$ and $E_2 \models \kappa_2$	(Sat-par)
E	$\models$	$\kappa_1 \wedge \kappa_2$	iff	$E \models \kappa_1$ and $E \models \kappa_2$	(Sat-and)
E	$\models$	$n[\kappa]$	iff	exists E' such that $E = n[E']$ and $E' \models \kappa$	(Sat-amb)
E	$\models$	$\mathtt{new}(n, \kappa)$	iff	exists E' such that $E = (\nu n)\ E'$ and $E' \models \kappa$	(Sat-new)
E	$\models$	$\oplus\kappa$	iff	exists E' such that $E \downharpoonleft\!\downharpoonright E'$ and $E' \models \kappa$	(Sat-next)
E	$\models$	$✧\kappa$	iff	exists E' such that $E \downharpoonleft\!\downharpoonright^* E'$ and $E' \models \kappa$	(Sat-sw)
E	$\models$	$\exists x.\kappa$	iff	exists n such that $E \models \kappa\{x \leftarrow n\}$	(Sat-exist)

Proof. Immediate from Eq. (4) and the equalities in Table 3. □

We let $\downharpoonleft\!\downharpoonright^*$ denote the reflexive and transitive closure of the spatial reduction relation $\downharpoonleft\!\downharpoonright$, i.e.

$$E_1 \downharpoonleft\!\downharpoonright^* E_2 \quad \text{iff} \quad E_1 = E_2 \text{ or exists } E_3 \text{ such that } E_1 \downharpoonleft\!\downharpoonright E_3 \text{ and } E_3 \downharpoonleft\!\downharpoonright^* E_2. \tag{5}$$

5. Context expressions

Based on the formal representation of contexts presented in the previous section, we define in this section a modal logic for specifying the properties of contexts. We call a formula in this logic a *Context Expression* (CE in short). The syntax and the formal semantics of CEs are given in Section 5.1. In Section 5.2, we show how CEs can be used to specify common context properties such as user location, nearby people and available resources.

5.1. Syntax and semantics

The syntax of CEs is given in Table 4 where κ ranges over CEs, n ranges over names and x is a variable symbol which also ranges over names.

The formal semantics of context expressions is given by the satisfaction relation $\models$ defined in Table 5, where E is universally quantified over the set of all contexts and the operator '$\downharpoonleft\!\downharpoonright^*$' is defined in Eq. (5). In Table 5 the notation $E \models \kappa$ states that the context E satisfies the context expression κ. We also denote by $\kappa\{x \leftarrow n\}$ the substitution of n for each free occurrence of x in κ. We now explain the meaning of context expressions. The CE **True** holds for all contexts described by the grammar in Table 2. It stands for the truth value *true*. A CE of the form $n = m$ holds for a context if the names n and m are lexically identical. This kind of CE is useful for checking whether two messages (names) are the same. For example suppose a process receives a message n from the network bandwidth sensor; the process can then take special measure to adapt depending on the value of n: *low*, *medium* or *high*. This mechanism can also be used to make a process be aware of other type of situations such as room temperature, noise level and lighting. The CE $\bullet$ holds solely for the hole context $\odot$.

This CE is particularly important as it denotes in a context expression 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 $\kappa_1 \mid \kappa_2$ holds for a context if that context is a composition of two contexts E_1 and E_2 such that κ_1 holds for E_1 and κ_2 holds for E_2. A CE $n[\kappa]$ holds for a context if that context is of the form $n[E]$ such that κ holds for the context E. A CE $\mathtt{new}(n, \kappa)$ holds for a context if that context has the form $(\nu n)\ E$ such that κ holds for the context E. A CE $\oplus\kappa$ holds for a context if that context can reduce in one step (with respect to the spatial reduction relation '$\downharpoonleft\!\downharpoonright$' defined in Eq. (4)) into

Table 6
Derived connectives.

False	$\widehat{=}$	$\neg$**True**	false
$\kappa_1 \vee \kappa_2$	$\widehat{=}$	$\neg(\neg\kappa_1 \wedge \neg\kappa_2)$	disjunction
$\kappa_1 \Rightarrow \kappa_2$	$\widehat{=}$	$\neg\kappa_1 \vee \kappa_2$	implication
$\kappa_1 \Leftrightarrow \kappa_2$	$\widehat{=}$	$(\kappa_1 \Rightarrow \kappa_2) \wedge (\kappa_2 \Rightarrow \kappa_1)$	logical equivalence

Table 7
Samples of context expressions.

$\mathtt{has}(n)$	$\widehat{=}$	$\oplus\ (\bullet \mid n[\mathbf{True}] \mid \mathbf{True})$	λ contains n
$\mathtt{at}(n)$	$\widehat{=}$	$n[\oplus(\bullet \mid \mathbf{True})] \mid \mathbf{True}$	λ is located at n
$\mathtt{with}(n)$	$\widehat{=}$	$n[\mathbf{True}] \mid \oplus(\bullet \mid \mathbf{True})$	λ is with n
$\mathtt{in\_with}(n)$	$\widehat{=}$	$\oplus\ (\bullet \mid \mathbf{True}) \mid \oplus(n[\mathbf{True}] \mid \mathbf{True})$	λ will be with n if λ moves in
$\mathtt{out\_with}(n)$	$\widehat{=}$	$n[\mathbf{True}] \mid \oplus\oplus\ (\bullet \mid \mathbf{True})$	λ will be with n if λ moves out
$\mathtt{out\_at}(n)$	$\widehat{=}$	$n[\oplus\oplus\ (\bullet \mid \mathbf{True})] \mid \mathbf{True}$	λ will be at n if λ moves out
$\mathtt{near}(n)$	$\widehat{=}$	$\mathtt{has}(n) \vee \mathtt{at}(n) \vee \mathtt{with}(n)$ $\vee\ \mathtt{in\_with}(n) \vee \mathtt{out\_with}(n)$ $\vee\ \mathtt{out\_at}(n)$	λ is nearby n
$\mathtt{at2}(n, m)$	$\widehat{=}$	$n[m[\mathbf{True}] \mid \mathbf{True}] \mid \mathbf{True}$	m is located at n
$\mathtt{with2}(n, m)$	$\widehat{=}$	$n[\mathbf{True}] \mid m[\mathbf{True}] \mid \mathbf{True}$	m is with n
$\mathtt{in\_with2}(n, m)$	$\widehat{=}$	$m[\mathbf{True}] \mid \oplus(n[\mathbf{True}] \mid \mathbf{True})$	m will be with n if m moves in
$\mathtt{out\_with2}(n, m)$	$\widehat{=}$	$\mathtt{in\_with2}(m, n)$	m will be with n if m moves out
$\mathtt{out\_at2}(n, m)$	$\widehat{=}$	$n[\oplus(m[\mathbf{True}] \mid \mathbf{True})] \mid \mathbf{True}$	m will be at n if m moves out
$\mathtt{near2}(n, m)$	$\widehat{=}$	$\mathtt{at2}(n, m) \vee \mathtt{out\_with2}(n, m)$ $\vee\ \mathtt{at2}(m, n) \vee \mathtt{in\_with2}(n, m)$ $\vee\ \mathtt{out\_at2}(n, m) \vee \mathtt{with2}(n, m)$	m is nearby n

a context for which κ holds. The operator $\oplus$ is called the *spatial next modality*. 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*.

Table 6 lists some derived connectives, illustrating properties that can be expressed in the logic. Their informal meaning can be understood in a usual manner as in classical predicate logic.

The following theorem is a fundamental property of the satisfaction relation $\models$; it states that satisfaction is invariant under equality of contexts. That is, logical formulae can only express properties that are invariant up to equality.

Theorem 5.1 *(Satisfaction is up to =).*

$$\big(E \models \kappa \textit{ and } E = E'\big) \quad \textit{implies} \quad E' \models \kappa.$$

The proof of this theorem can be found in Appendix B.1.

Definition 5.1 *(Validity).* A context expression κ is valid, and we write $\models \kappa$, if it holds for all contexts, i.e.

$$\models \kappa \quad \text{iff} \quad E \models \kappa, \quad \text{for all context } E.$$

5.2. Examples of context expressions

We now give some examples to show how context expressions can be used to specify the properties of the context of CCA processes. We define in Table 7 some 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; the parameters n and m are ambient names. There are two type of predicates: unary and binary. Each of the unary predicates expresses some property of context in relation to the ambient passed as parameter and the implicit ambient λ evaluating the predicates, while a binary predicate specifies some relationship between the two ambients passed as parameters in the execution context of the implicit ambient λ evaluating that predicate. This set of predicates is an attempt towards the creation of a user-friendly interface for non-specialist users to specify the context of their applications. The informal meaning of these predicates is explained below.

1) The context expression $\mathtt{has}(n)$ holds if the ambient λ evaluating the expression has no parent and contains an ambient named n. For example, $\mathtt{has}(pda)$ holds for the context $bob[\odot \mid pda[\mathbf{0}]]$ and we write

$$bob\big[\odot \mid pda[\mathbf{0}]\big] \models \mathtt{has}(pda). \tag{6}$$

Here the ambient evaluating the CE is bob, i.e. $\lambda = bob$; the ambient bob has no parent and contains an ambient named pda. The formal proof of Eq. (6) is given by Table 8, based on the algebraic semantics of context defined in Table 3, the satisfaction relation defined in Table 5 and Theorem 5.1. Note that the CE $\mathtt{has}(pda)$ does not hold for the context

Table 8
Formal proof of Eq. (6).

1.	$\mathbf{0} \models \mathbf{True}$	{(Sat-true)}
2.	$pda[\mathbf{0}] \models pda[\mathbf{True}]$	{1 and (Sat-amb)}
3.	$pda[\mathbf{0}] \mid \mathbf{0} \models pda[\mathbf{True}] \mid \mathbf{True}$	{1, 2 and (Sat-par)}
4.	$\odot \models \bullet$	{(Sat-hole)}
5.	$\odot \mid pda[\mathbf{0}] \mid \mathbf{0} \models \bullet \mid pda[\mathbf{True}] \mid \mathbf{True}$	{3, 4 and (Sat-par)}
6.	$(\odot \mid pda[\mathbf{0}] \mid \mathbf{0}) = (\odot \mid pda[\mathbf{0}])$	{(Cont-0)}
7.	$\odot \mid pda[\mathbf{0}] \models \bullet \mid pda[\mathbf{True}] \mid \mathbf{True}$	{5, 6 and Theorem 5.1}
8.	$bob[\odot \mid pda[\mathbf{0}]] \models \oplus\ (\bullet \mid pda[\mathbf{True}] \mid \mathbf{True})$	{7 and (Sat-next)}

$conf[bob[\odot \mid pda[\mathbf{0}]]]$ because the evaluating ambient *bob* has a parent which is the ambient *conf*; viz

$$conf\big[bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \not\models \mathtt{has}(pda).$$

But using the spatial operator $\oplus$ to move one step down in that hierarchy, we have

$$conf\big[bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \oplus\,\mathtt{has}(pda).$$

Similarly, the CE $\diamond\mathtt{has}(pda)$ holds for any context that has somewhere the context $bob[\odot \mid pda[\mathbf{0}]]$ as sub-context.

2) The context expression $\mathtt{at}(n)$ holds for a context if the ambient λ evaluating the expression is located in a root ambient named n. For example, we have

$$conf\big[bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \mathtt{at}(conf).$$

This can be formally proved in a similar manner as Eq. (6).

3) The context expression $\mathtt{with}(n)$ holds for a context if λ and n are both roots in that context. For example, we have

$$alice[\mathbf{0}] \mid bob\big[\odot \mid pda[\mathbf{0}]\big] \models \mathtt{with}(alice).$$

We also have

$$conf\big[alice[\mathbf{0}] \mid bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \mathtt{at}(conf) \wedge \oplus\mathtt{with}(alice),$$

meaning that in the context on the left of '$\models$', λ which is *bob* here is at the conference room with *alice*.

4) The context expression $\mathtt{in\_with}(n)$ holds for a context if λ and the parent of n are roots in that context. This means that if λ performs the capability '$\mathtt{in}$' to move into n's parent, then λ will be co-located with n; hence the name given to this predicate. For example we have

$$conf\big[alice[\mathbf{0}]\big] \mid bob\big[\odot \mid pda[\mathbf{0}]\big] \models \mathtt{in\_with}(alice).$$

So if *bob* moves into the conference room, it will be co-located with *alice*.

5) The context expression $\mathtt{out\_with}(n)$ holds for a context if λ's parent and n are roots in that context. Thus if λ performs the capability '$\mathtt{out}$' to move out of its parent, then λ will be co-located with n. For example we have

$$alice[\mathbf{0}] \mid conf\big[bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \mathtt{out\_with}(alice).$$

So if *bob* moves out of the conference room, it will be co-located with *alice*.

6) The context expression $\mathtt{out\_at}(n)$ holds for a context if n is a root and is λ's grand-parent in that context. Therefore, if λ performs the capability '$\mathtt{out}$' to move out of its parent, then λ will be located at n. For example we have

$$campus\big[conf\big[bob\big[\odot \mid pda[\mathbf{0}]\big]\big]\big] \models \mathtt{out\_at}(campus).$$

So if *bob* moves out of the conference room, it will find itself in the *campus* outside the conference room.
We can now give the formal definitions of the context expressions $\mathtt{user\_at}(n)$ and $\mathtt{user\_with}(n)$ used in Section 2.1. The context expression $\mathtt{user\_at}(n)$ holds for a context if the parent of the ambient evaluating it is located at n in that context (regardless of the position where this situation occurs in that context). So we write

$$\mathtt{user\_at}(n) \mathrel{\widehat{=}} \diamond\mathtt{out\_at}(n). \tag{7}$$

The context expression $\mathtt{user\_with}(n)$ holds for a context if the parent of the ambient evaluating it is co-located with n in that context (regardless the position where this situation occurs in that context). So we write

$$\mathtt{user\_with}(n) \mathrel{\widehat{=}} \diamond\mathtt{out\_with}(n). \tag{8}$$

7) The context expression $\mathtt{near}(n)$ holds for a context if n is nearby λ in that context. For the sake of simplicity, the vicinity of λ is limited here to its children, its siblings and their children, its grand-parent, and its grand-parent's children. However, what is considered to be nearby depends on application domains and can be defined in a similar manner.

Table 9
Structural congruence for processes.

(S1)	$P \equiv P$	(S12)	$(\nu n)\ P \mid Q \equiv (\nu n)\ (P \mid Q)$ if $n \notin \mathbf{fn}(Q)$
(S2)	$P \equiv Q \quad \Rightarrow \quad Q \equiv P$	(S13)	$(\nu n)\ m[P] \equiv m[(\nu n)\ P]$ if $n \neq m$
(S3)	$P \equiv Q,\ Q \equiv R \quad \Rightarrow \quad P \equiv R$	(S14)	$!P \equiv P \mid !P$
(S4)	$P \equiv Q \quad \Rightarrow \quad (\nu n)P \equiv (\nu n)\ Q$	(S15)	$P \mid \mathbf{0} \equiv P$
(S5)	$P \equiv Q \quad \Rightarrow \quad P \mid R \equiv Q \mid R$	(S16)	$!\mathbf{0} \equiv \mathbf{0}$
(S6)	$P \equiv Q \quad \Rightarrow \quad !P \equiv !Q$	(S17)	$\mathbf{0}.P \equiv P$
(S7)	$P \equiv Q \quad \Rightarrow \quad n[P] \equiv n[Q]$	(S18)	$(\nu n)\ \mathbf{0} \equiv \mathbf{0}$
(S8)	$P \equiv Q \quad \Rightarrow \quad M.P \equiv M.Q$	(S19)	$\mathbf{True}?M.P \equiv M.P$
(S9)	$P \mid Q \equiv Q \mid P$	(S20)	$P \equiv Q,\ (\models \kappa \Leftrightarrow \kappa') \quad \Rightarrow \quad \kappa?P \equiv \kappa'?Q$
(S10)	$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	(S21)	$P \equiv Q \quad \Rightarrow \quad x \triangleright (\tilde{y}).P \equiv x \triangleright (\tilde{y}).Q$
(S11)	$(\nu n)\ (\nu m)\ P \equiv (\nu m)\ (\nu n)\ P$	(S22)	$\mathtt{del}\ n.P \mid n[\mathbf{0}] \equiv P$

8) The meaning of a binary predicate '$X2(n,m)$' is similar to that of the corresponding unary predicate '$X(n)$' where the implicit parameter λ is explicitly represented by m. Unlike λ, m can be any ambient, not only the one evaluating the predicate. For example, the predicate $\mathtt{at2}(n,m)$ holds for a context if the ambient m is located at a root ambient named n in that context. This is illustrated by the following examples:

$$conf\big[alice[\mathbf{0}] \mid bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \mathtt{at2}(conf, alice)$$
$$conf\big[alice[\mathbf{0}] \mid bob\big[\odot \mid pda[\mathbf{0}]\big]\big] \models \mathtt{at2}(conf, bob)$$

meaning that *alice* and *bob* are located at *conf* in the context under consideration, respectively. The other binary predicates can be explained in a similar way.

6. A reduction semantics for CCA

The operational semantics of CCA is defined using a structural congruence $\equiv$ and a reduction relation $\rightarrow$. The structural congruence is the smallest congruence relation on processes that satisfies the axioms in Table 9. These axioms allow the manipulation of the structure of processes. It follows from these axioms that the structural congruence is a commutative monoid for $(\mathbf{0}, \mid)$. These axioms are inherited from MA [3] except the axioms (S19)–(S22). The axiom (S19) says that a capability guarded with **True** is the same as that capability without guards. The next two axioms (S20) and (S21) define the equivalence of context-guarded processes and process abstractions respectively. Finally the axiom (S22) states that an ambient that has completed its computation can be explicitly deleted from the system using the capability del.

The reduction relation of processes is defined in Table 10. We write $P\{\tilde{y} \leftarrow \tilde{z}\}$ for the substitution of each name in the list $\tilde{z}$ for each free occurrence of the corresponding name in the list $\tilde{y}$. Such a substitution is only defined if $|\tilde{y}| = |\tilde{z}|$. The first set of rules (Red Call) gives the semantics of a process call. It states that a process call takes place only if a corresponding process abstraction is defined at the specified location. The next set of rules (Red Com) are related to message passing communication between processes and across ambient boundaries. The mobility rules (Red In) and (Red Out) are inherited from MA. The rules (Red Res) to (Red $\equiv$) propagate reduction across scopes, ambient nesting and parallel composition; and ensure that the reduction relation is closed under structural equivalence respectively.

The last two rules link the context model presented in Section 4 to the reduction relation. The rule (Red Cont) says that a process in the *hole* of a context can reduce at any time while the rule (Red Guard) asserts that a context-guarded capability reduces in a context if that context satisfies the guard of that capability. In the rule (Red Guard) the reduction $E(M.P) \rightarrow E'(P)$ and not $E(M.P) \rightarrow E(P)$ takes into account the fact that the capability M may be a mobility one; in which case the context E will change to E' where the ambient performing the capability M has changed its location. For example suppose $E \mathrel{\widehat{=}} Q \mid n[a[\odot \mid R] \mid b[S]]$, for some processes Q, R and S, and some ambient names n, a and b. We have the following reduction:

$$E(\mathtt{in}\, b.P) \equiv \big(Q \mid n\big[a[\mathtt{in}\, b.P \mid R] \mid b[S]\big]\big) \quad \rightarrow \quad \big(Q \mid n\big[b\big[a[P \mid R] \mid S\big]\big]\big) \equiv E'(P)$$

where $E' \mathrel{\widehat{=}} Q \mid n[b[a[\odot \mid R] \mid S]]$. In E', the ambient a has moved into the ambient b.

7. Application to health care

In this section, we specify in CCA the *context-aware hospital bed* designed in the *Hospital of the Future* project at the Centre for Pervasive Health Care, Denmark [11,12]. This example is chosen because it represents a real-world application of context-aware computing in health care. The bed has an integrated computer and a touch sensitive display which is used by the patients for entertainment purposes (e.g. for watching television) and the clinicians for accessing medical data while working at the bed. The bed is aware of who is using it (i.e. the identity of the patient), and what and who is near it. For example, the bed is aware of the nurse, the patient and the medicine tray. The bed runs a context-aware EPR (Electronic Patient Record) client. Based on the location of the nurse, the patient and the medicine tray, the bed can automatically log in the nurse, find the patient record, display the medicine schema, and in this schema highlight the prescribed medicine

Table 10
Reduction relation for processes.

$x \triangleright (\tilde{y}).P \mid x\langle\tilde{z}\rangle \quad \rightarrow \quad x \triangleright (\tilde{y}).P \mid P\{\tilde{y} \leftarrow \tilde{z}\}$	(Red Call Lc)
$n[Q \mid x \triangleright (\tilde{y}).P] \mid m[:: x\langle\tilde{z}\rangle \mid R] \quad \rightarrow \quad n[Q \mid x \triangleright (\tilde{y}).P] \mid m[P\{\tilde{y} \leftarrow \tilde{z}\} \mid R]$	(Red Call S1)
$n[Q \mid x \triangleright (\tilde{y}).P] \mid m[n :: x\langle\tilde{z}\rangle \mid R] \quad \rightarrow \quad n[Q \mid x \triangleright (\tilde{y}).P] \mid m[P\{\tilde{y} \leftarrow \tilde{z}\} \mid R]$	(Red Call S2)
$Q \mid x \triangleright (\tilde{y}).P \mid m[\uparrow x\langle\tilde{z}\rangle \mid R] \quad \rightarrow \quad Q \mid x \triangleright (\tilde{y}).P \mid m[P\{\tilde{y} \leftarrow \tilde{z}\} \mid R]$	(Red Call U1)
$n[Q \mid x \triangleright (\tilde{y}).P \mid m[n\uparrow x\langle\tilde{z}\rangle \mid R]] \quad \rightarrow \quad n[Q \mid x \triangleright (\tilde{y}).P \mid m[P\{\tilde{y} \leftarrow \tilde{z}\} \mid R]]$	(Red Call U2)
$Q \mid\downarrow x\langle\tilde{z}\rangle \mid m[R \mid x \triangleright (\tilde{y}).P] \quad \rightarrow \quad Q \mid P\{\tilde{y} \leftarrow \tilde{z}\} \mid m[R \mid x \triangleright (\tilde{y}).P]$	(Red Call D1)
$Q \mid m\downarrow x\langle\tilde{z}\rangle \mid m[R \mid x \triangleright (\tilde{y}).P] \quad \rightarrow \quad Q \mid P\{\tilde{y} \leftarrow \tilde{z}\} \mid m[R \mid x \triangleright (\tilde{y}).P]$	(Red Call D2)
$(\tilde{y}).P \mid \langle\tilde{z}\rangle.Q \quad \rightarrow \quad P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q$	(Red Com Lc)
$n[:: (\tilde{y}).P \mid Q] \mid m[:: \langle\tilde{z}\rangle.R \mid S] \quad \rightarrow \quad n[P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q] \mid m[R \mid S]$	(Red Com S1)
$n[:: (\tilde{y}).P \mid Q] \mid m[n :: \langle\tilde{z}\rangle.R \mid S] \quad \rightarrow \quad n[P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q] \mid m[R \mid S]$	(Red Com S2)
$n[m :: (\tilde{y}).P \mid Q] \mid m[:: \langle\tilde{z}\rangle.R \mid S] \quad \rightarrow \quad n[P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q] \mid m[R \mid S]$	(Red Com S3)
$n[m :: (\tilde{y}).P \mid Q] \mid m[n :: \langle\tilde{z}\rangle.R \mid S] \quad \rightarrow \quad n[P\{\tilde{y} \leftarrow \tilde{z}\} \mid Q] \mid m[R \mid S]$	(Red Com S4)
$\downarrow(\tilde{y}).P \mid m[\uparrow\langle\tilde{z}\rangle.Q \mid R] \quad \rightarrow \quad P\{\tilde{y} \leftarrow \tilde{z}\} \mid m[Q \mid R]$	(Red Com R1)
$n\downarrow(\tilde{y}).P \mid n[\uparrow\langle\tilde{z}\rangle.Q \mid R] \quad \rightarrow \quad P\{\tilde{y} \leftarrow \tilde{z}\} \mid n[Q \mid R]$	(Red Com R2)
$n\downarrow\langle\tilde{z}\rangle.P \mid n[\uparrow(\tilde{y}).Q \mid R] \quad \rightarrow \quad P \mid n[Q\{\tilde{y} \leftarrow \tilde{z}\} \mid R]$	(Red Com R3)
$\downarrow\langle\tilde{z}\rangle.P \mid m[\uparrow(\tilde{y}).Q \mid R] \quad \rightarrow \quad P \mid m[Q\{\tilde{y} \leftarrow \tilde{z}\} \mid R]$	(Red Com R4)
$u[\downarrow(\tilde{y}).P \mid m[u\uparrow\langle\tilde{z}\rangle.Q \mid R] \mid S] \quad \rightarrow \quad u[P\{\tilde{y} \leftarrow \tilde{z}\} \mid m[Q \mid R] \mid S]$	(Red Com R5)
$u[n\downarrow(\tilde{y}).P \mid n[u\uparrow\langle\tilde{z}\rangle.Q \mid R] \mid S] \quad \rightarrow \quad u[P\{\tilde{y} \leftarrow \tilde{z}\} \mid n[Q \mid R] \mid S]$	(Red Com R6)
$u[n\downarrow\langle\tilde{z}\rangle.P \mid n[u\uparrow(\tilde{y}).Q \mid R] \mid S] \quad \rightarrow \quad u[P \mid n[Q\{\tilde{y} \leftarrow \tilde{z}\} \mid R] \mid S]$	(Red Com R7)
$u[\downarrow\langle\tilde{z}\rangle.P \mid m[u\uparrow(\tilde{y}).Q \mid R] \mid S] \quad \rightarrow \quad u[P \mid m[Q\{\tilde{y} \leftarrow \tilde{z}\} \mid R] \mid S]$	(Red Com R8)
$n[\texttt{in}\ m.P \mid Q] \mid m[R] \quad \rightarrow \quad m[n[P \mid Q] \mid R]$	(Red In)
$m[n[\texttt{out}.P \mid Q] \mid R] \quad \rightarrow \quad n[P \mid Q] \mid m[R]$	(Red Out)
$P \rightarrow P' \quad \Rightarrow \quad (\nu n)\ P \rightarrow (\nu n)\ P'$	(Red Res)
$P \rightarrow P' \quad \Rightarrow \quad n[P] \rightarrow n[P']$	(Red Amb)
$P \rightarrow P' \quad \Rightarrow \quad P \mid Q \rightarrow P' \mid Q$	(Red Par)
$P \equiv Q,\ Q \rightarrow Q',\ Q' \equiv P' \quad \Rightarrow \quad P \rightarrow P'$	(Red $\equiv$)
$P \rightarrow P' \quad \Rightarrow \quad E(P) \rightarrow E(P')$	(Red Cont)
$E(M.P) \rightarrow E'(P) \quad \Rightarrow \quad E(\kappa?M.P) \rightarrow E'(P) \quad \text{if } E \models \kappa$	(Red Guard)
$\rightarrow^*$ reflexive and transitive closure of $\rightarrow$	

which is in the pill container. The nurse is automatically logged out of the computer when she leaves the active zone of the bed. This mechanism of logging in and logging out a user based on its proximity is called *proximity-based user authentication* [11,12]. The notion of *active zone* is used in the prototype context-aware bed to delimit the range of the bed awareness. The bed is aware of changes that occur within its active zone; for example when a nurse enters or leaves that zone.

As discussed in [12], context is more than location in a hospital setting. For example, the nurse documenting the medicine needs not be located close to the patient, even though this patient is definitely part of her work context. Moreover, the patient EPR may be stored on a remote server but still it provides valuable context information about the patient condition and treatment.

We consider six entities: the bed, the patient, the nurse, the medicine tray, the pill container and the active zone. Each of these entities is represented by an ambient in CCA. For simplicity we assume the patient is male and the nurse is female. Initially, the bed is located in its active zone, the patient is inside the bed, and the nurse and medicine tray containing the pill containers are outside the active zone of the bed as depicted in Fig. 1. This is modelled by the following process:

$$\begin{array}{lr} \quad \mathit{nurse}[P_{\mathrm{n}}] & \text{(nurse ambient)} \\ \mid \mathit{tray}\big[P_{\mathrm{t}} \mid \mathit{con}_1[P_1] \mid \cdots \mid \mathit{con}_k[P_k]\big] & \text{(tray ambient with } k \text{ pill containers)} \\ \mid \mathit{zone}\big[\mathit{bed}\big[P_{\mathrm{b}} \mid \mathit{patient\_001}[P_{\mathrm{p}}]\big]\big] & \text{(active zone, bed and patient ambients),} \end{array}$$

where P_{X} is a process specifying the behaviour of its host ambient.

Nurse ambient. The nurse can enter and leave the active zone as often as needed in the course of her work activities. This is done by the corresponding ambient performing the capability '$\texttt{in}\ \mathit{zone}$' to enter the zone and the capability $\texttt{out}$ to leave the zone. Once in the active zone, the nurse can access the patient EPR using the touch screen embedded in the bed. We assume that the nurse bring with her the medicine tray in and out the active zone. This requires a synchronisation between the ambient representing the nurse and the ambient modelling the medicine tray. Such a synchronisation is modelled using handshake message passing communication primitives. So the behaviour of the nurse can be modelled as follows:

$$\begin{array}{lll} P_{\mathrm{n}} \mathrel{\widehat{=}} & !\,\texttt{with}(\mathit{zone})?\mathit{tray} :: \langle\rangle.\texttt{in}\ \mathit{zone}.\mathbf{0} & (a) \\ & \mid\ !\,\texttt{at}(\mathit{zone})?\mathit{bed} :: \texttt{epr}\langle\rangle.\mathbf{0} & (b) \\ & \mid\ !\,\texttt{at}(\mathit{zone})?\mathit{tray} :: \langle\rangle.\texttt{out}.\mathbf{0} & (c) \end{array} \tag{9}$$

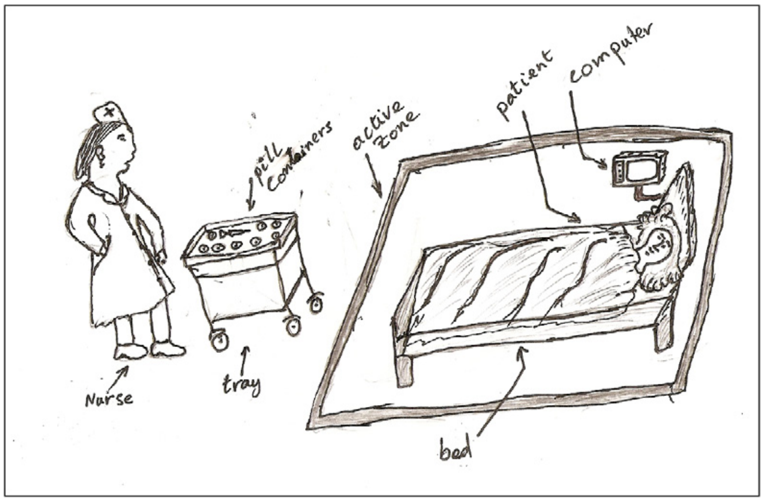

Fig. 1. Context-aware hospital bed system.

The process in Eq. (9)-*a* says that when the nurse ambient is next to the active zone (see the context expression '`with`(*zone*)') and is willing to enter the zone, it sends a message to the tray ambient to signal its intention to move in the active zone. When the tray ambient responds by receiving this message, the nurse ambient enters the zone by performing the capability '`in` *zone*'. Once in the active zone (see the context expression '`at`(*zone*)'), the nurse ambient can access the patient record by calling the process abstraction `epr` located in the bed ambient. This is modelled by the process in Eq. (9)-*b*. The process in Eq. (9)-*c* says that the nurse ambient signals its intention to leave the active zone to the tray ambient by sending a message to it. At the receipt of the message, the nurse ambient leaves the zone by performing the capability `out`.

Tray ambient. It is assumed that the tray ambient follows the nurse ambient in and out the active zone to supply patient medicine stored in pill containers. So the tray ambient communicates with the nurse ambient to know when to move in and out the active zone. This is modelled as follows and the explanation is similar to that of the nurse ambient

$$\begin{aligned} P_{\mathrm{t}} \;\widehat{=}\;\; & !\,\texttt{with}(zone)?nurse :: ().\texttt{in}\,zone.\mathbf{0} \\ & \mid\, !\,\texttt{at}(zone)?nurse :: ().\texttt{out}.\mathbf{0} \end{aligned}$$

Pill container ambient. A pill container is aware of the patient and reveals itself when near the patient by lighting the name of the patient. For example, if the pill container con_i, for some i such that $1 \leqslant i \leqslant k$, contains the medicine of the patient named *patient_001*, then the behaviour of the pill container is specified as follows:

$$P_i \;\widehat{=}\;\; !\,\diamond\!\texttt{near}(patient\_001)?\uparrow\langle patient\_001\rangle.\mathbf{0}$$

So the pill container con_i lights the patient name by sending the name to its parent ambient, which will eventually propagate the message. This is how we model the lighting of names here. So we will change P_{t} as follows to enable communication between the tray ambient and the pill container ambients:

$$\begin{aligned} P_{\mathrm{t}} \;\widehat{=}\;\; & !\,\texttt{with}(zone)?nurse :: ().\texttt{in}\,zone.\mathbf{0} \\ & \mid\, !\,\texttt{at}(zone)?nurse :: ().\texttt{out}.\mathbf{0} \\ & \mid\, !\downarrow(msg).::\langle msg\rangle.\mathbf{0} \end{aligned}$$

The tray ambient is now able to receive messages from the pill container ambients it contains (by performing the capability '$\downarrow(msg)$') and forward them to the ambients around it such as the nurse ambient and the bed ambient (by performing the capability '$::\langle msg\rangle$').

Patient ambient. The patient ambient selects an entertainment program by sending a request to the bed ambient as follows:

$$P_{\mathrm{p}} \;\widehat{=}\;\; !\,\nu req\; bed\uparrow\langle req\rangle.\mathbf{0}$$

The restriction operator 'ν' models here the *freshness* of a request.

Bed ambient. The bed ambient is located in its active zone which delimits the context of the bed. One of the most important context-awareness properties of the bed is the ability of logging the nurse in when she enters the active zone and logging her out when she leaves that zone, automatically. This is modelled as follows:

$$\begin{aligned} P_{\mathrm{b}} \mathrel{\widehat{=}}\ & (\diamondsuit\mathtt{at2}(zone, nurse))?\mathtt{login}\langle nurse\rangle.\mathbf{0} && (a) \\ & \mid\ !\,(\neg\diamondsuit\mathtt{at2}(zone, nurse))?\mathtt{logout}\langle\rangle.(\diamondsuit\mathtt{at2}(zone, nurse))?\mathtt{login}\langle nurse\rangle.\mathbf{0} && (b) \end{aligned} \qquad (10)$$

The context-guarded capability in Eq. (10)-*a* says that when the nurse enter the bed's active zone, the action '$\mathtt{login}\langle nurse\rangle$' is taken to log the nurse in the EPR system. The nurse is logged out when she leaves the active zone as specified in Eq. (10)-*b*. The replication operator '!' in Eq. (10)-*b* means that the sequence 'Eq. (10)-*a* followed by Eq. (10)-*b*' is repeated forever. The process abstractions `login`, `logout` and `epr` are detailed later in Section 9.4, after the presentation of specific data structures used in their specifications.

8. A theory of equivalence of context-aware systems

In this section, we develop a theory of contextual equivalence for proving properties of context-aware systems. Our approach is based on *Morris-style contextual equivalence* [4] which is a standard way of testing that two processes are equivalent and stipulates that:

> two processes are contextually equivalent if and only if they admit the same elementary observations whenever they are inserted inside any arbitrary enclosing process.

This technique is also known as *may-testing equivalence* in [13].

In CCA, we define the elementary observations to be the presence, at the top-level of a process, of an ambient which is aware of specific contexts and whose name is not restricted. As customary, an arbitrary enclosing process is modelled by the notion of *generalised context*. A generalised context is a process with one or more holes in it. This generalises the notion of context defined in Section 4 where no more than one hole is allowed in a context. We let C denote a generalised context and $C(P)$ represent the process obtained by replacing each hole in C with a copy of the process P. Note that as a result of such a substitution, names free in P may become bound in $C(P)$. So, contexts are not identified up to renaming of bound names. The contextual equivalence relation is defined in terms of the *observability predicate* and the *convergence predicate* which are defined below.

Definition 8.1. We say that an ambient named n is aware of the contexts described by a context expression κ if that ambient contains a process guarded by the context expression κ, i.e. that ambient has the form

$$n\big[(\kappa?P) \mid Q\big],$$

for some processes P and Q.

Definition 8.2 *(Observability predicate).* For each ambient name n and each context expression κ, the *observability predicate* $\perp_n^\kappa$ is defined over processes by:

> $P\perp_n^\kappa$ if and only if the process P contains a top-level ambient named n which is aware of the contexts that satisfy the context expression κ, and the name n is not restricted. That is
>
> $$P\perp_n^\kappa \quad \text{iff} \quad P \equiv (\nu n_1, \ldots, n_i)\ \big(n[\kappa?Q \mid R] \mid S\big),$$
>
> for some names $n_1, \ldots, n_i$ such that $n \notin \{n_1, \ldots, n_i\}$, and some processes Q, R and S.

Example 8.1. The expression

$$phone\big[\mathtt{at}(conf)?\mathtt{switchto}\langle silent\rangle.\mathbf{0}\big]\ \perp_{phone}^{\mathtt{at}(conf)}$$

is true because *phone* is a top-level ambient, non-restricted and aware of the contexts described by the context expression $\mathtt{at}(conf)$.

Example 8.2. The expression

$$bob\big[phone\big[\mathtt{at}(conf)?\mathtt{switchto}\langle silent\rangle.\mathbf{0}\big]\big]\ \perp_{phone}^{\mathtt{at}(conf)}$$

is false because *phone* is not a top-level ambient.

Example 8.3. The expression

$$(\nu\ phone)\ phone\big[\mathtt{at}(conf)?\mathtt{switchto}\langle silent\rangle.\mathbf{0}\big]\ \perp_{phone}^{\mathtt{at}(conf)}$$

is false because the name *phone* is restricted.

Definition 8.3 *(Convergence predicate).* For each ambient name n and each context expression κ, the *convergence predicate* $\perp\!\!\!\perp_n^\kappa$ is defined over processes by:

$P \perp\!\!\!\perp_n^\kappa$ if and only if the process P can reduce in a finite number of steps into a process that contains a top-level ambient named n which is aware of the contexts described by the context expression κ, and the name n is not restricted. That is

$$P \perp\!\!\!\perp_n^\kappa \quad \text{iff} \quad P \perp_n^\kappa \text{ or } \big(P \to P' \text{ and } P' \perp\!\!\!\perp_n^\kappa\big),$$

for some process P'.

Example 8.4. The expression

$$\mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.0\big] \perp\!\!\!\perp_{\mathit{phone}}^{\mathtt{at}(\mathit{conf})}$$

is true because *phone* is a top-level ambient, non-restricted and aware of the contexts described by the context expression $\mathtt{at}(\mathit{conf})$.

Example 8.5. The expression

$$\mathit{bob}\big[\mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.\mathbf{0} \mid \mathtt{out}.\mathbf{0}\big]\big] \perp\!\!\!\perp_{\mathit{phone}}^{\mathtt{at}(\mathit{conf})}$$

is true because of the following reasons:

1. the ambient *phone* can perform the capability out to move out of the ambient *bob*, as illustrated by the following derivation:

$$\mathit{bob}\big[\mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.\mathbf{0} \mid \mathtt{out}.\mathbf{0}\big]\big] \quad \to \quad \mathit{bob}[\mathbf{0}] \mid \mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.\mathbf{0}\big]$$

2. following this derivation, the ambient *phone* becomes a top-level ambient, not restricted and aware of the contexts described by the context expression $\mathtt{at}(\mathit{conf})$, viz.

$$\mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.\mathbf{0}\big] \perp_{\mathit{phone}}^{\mathtt{at}(\mathit{conf})}.$$

Example 8.6. The expression

$$(\nu\ \mathit{phone})\ \mathit{phone}\big[\mathtt{at}(\mathit{conf})?\mathtt{switchto}\langle \mathit{silent}\rangle.0\big] \perp\!\!\!\perp_{\mathit{phone}}^{\mathtt{at}(\mathit{conf})}$$

is false because the name *phone* is restricted.

Definition 8.4 *(Contextual equivalence $\simeq$).* Two processes P and Q are contextually equivalent if and only if they admit the same elementary observations in all generalised context, i.e.

$$P \simeq Q \quad \text{iff} \quad C(P) \perp\!\!\!\perp_n^\kappa \quad \Leftrightarrow \quad C(Q) \perp\!\!\!\perp_n^\kappa,$$

for all ambient name n, all context expression κ and all generalised context C.

Example 8.7. If $n \neq m$ then $n[\kappa?().\mathbf{0}] \not\simeq m[\kappa?().\mathbf{0}]$.

Proof. We just have to find a generalised context C that distinguishes the two processes $n[\kappa?().\mathbf{0}]$ and $m[\kappa?().\mathbf{0}]$. Let $C = \odot$, i.e. the hole context. Then $C(n[\kappa?().\mathbf{0}]) \equiv n[\kappa?().\mathbf{0}]$, and so from Definition 8.2 we have $C(n[\kappa?().\mathbf{0}]) \perp_n^\kappa$. Then $C(n[\kappa?().\mathbf{0}]) \perp\!\!\!\perp_n^\kappa$ follows from Definition 8.3. Note that there is no reductions for the process $C(m[\kappa?().\mathbf{0}]) \equiv m[\kappa?().\mathbf{0}]$ because the input command $()$ can only take place if there is another process in the ambient m which is willing to perform an output $\langle\rangle$. So because $n \neq m$, $C(m[\kappa?().\mathbf{0}]) \perp\!\!\!\perp_n^\kappa$ is false. □

Example 8.8. If $\kappa_1 \not\Leftrightarrow \kappa_2$ then $\kappa_1?().\mathbf{0} \not\simeq \kappa_2?().\mathbf{0}$.

Proof. Let us consider the context $C = n[\odot]$. So, $C(\kappa_1?().\mathbf{0}) \equiv n[\kappa_1?().\mathbf{0}]$ and $C(\kappa_2?().\mathbf{0}) \equiv n[\kappa_2?().\mathbf{0}]$. For the same reasons stated in the proof of Example 8.7, there is no reductions for $C(\kappa_1?().\mathbf{0})$, nor $C(\kappa_2?().\mathbf{0})$. So we have $C(\kappa_1?().\mathbf{0}) \perp\!\!\!\perp_n^{\kappa_1}$. Because $\kappa_1 \not\Leftrightarrow \kappa_2$, it follows from the rule (S20) in Table 9 that $\kappa_1?().\mathbf{0} \not\equiv \kappa_2?().\mathbf{0}$, and then from the rule (S7) of the same table that $n[\kappa_1?().\mathbf{0}] \not\equiv n[\kappa_2?().\mathbf{0}]$. Consequently, $C(\kappa_2?().\mathbf{0}) \perp\!\!\!\perp_n^{\kappa_1}$ is false. □

Example 8.9. $\mathtt{in}\ n.\mathbf{0} \not\simeq \mathtt{out}.\mathbf{0}$.

Proof. The two processes are distinguishable by the context $C = m[r[\odot] \mid n[\mathbf{0}]]$ because $C(\texttt{in}\ n.\mathbf{0})\perp\!\!\!\perp_r^{\mathbf{True}}$ is false and $C(\texttt{out}.\mathbf{0})\perp\!\!\!\perp_r^{\mathbf{True}}$ is true as shown below.

- The only possible derivation of $C(\texttt{in}\ n.\mathbf{0})$ is

$$C(\texttt{in}\,n.\mathbf{0}) \equiv m\big[r[\texttt{in}\,n.\mathbf{0}] \mid n[\mathbf{0}]\big] \quad \rightarrow \quad m\big[n\big[r[\mathbf{0}]\big]\big]$$

  It follows that $C(\texttt{in}\ n.\mathbf{0})\perp\!\!\!\perp_r^{\mathbf{True}}$ is false because r is never a top-level ambient.
- The only possible derivation of $C(\texttt{out}.\mathbf{0})$ is the following:

$$C(\texttt{out}.\mathbf{0}) \equiv m\big[r[\texttt{out}.\mathbf{0}] \mid n[\mathbf{0}]\big] \quad \rightarrow \quad m\big[n[\mathbf{0}]\big] \mid r[\mathbf{0}]$$

  It follows that $C(\texttt{out}.\mathbf{0})\perp\!\!\!\perp_r^{\mathbf{True}}$ is true because r can become a top-level ambient. □

The first important property of the contextual equivalence $\simeq$ is given by the following theorem.

Theorem 8.1. *Contextual equivalence is a congruence relation.*

The proof of this theorem is given in Appendix B.2.

The second important property of the contextual equivalence is given by the following theorem which says that the structural congruence $\equiv$ is a subset of the contextual equivalence $\simeq$.

Theorem 8.2. *If $P \equiv Q$ then $P \simeq Q$.*

Proof. Let P and Q be two processes such that $P \equiv Q$. Suppose $C(P)\perp\!\!\!\perp_n^{\kappa}$ is true for some name n, context expression κ and generalised context C. Because $\equiv$ is a congruence (see Section 6), we have $C(P) \equiv C(Q)$. Therefore, from Definition 8.3 it follows that $C(Q)\perp\!\!\!\perp_n^{\kappa}$. We just proved that $C(P)\perp\!\!\!\perp_n^{\kappa} \Rightarrow C(Q)\perp\!\!\!\perp_n^{\kappa}$. The reverse is done in the similar way. We then conclude that $P \simeq Q$. □

9. Expressiveness of CCA

In this section we discuss the expressiveness of CCA in comparison with the π-calculus which is known to be a universal model of computation [5–7]. It is also known that the asynchronous π-calculus can encode any process of the original π-calculus [6,7]. We show that the asynchronous π-calculus can be encoded in CCA (Section 9.1). In addition, we illustrate how common data structures such a buffer (Section 9.2) and a persistent memory cell (Section 9.3) can be modelled in CCA in a natural manner. These data structures can be used for storing context information such as user preferences, device profiles and even the history of context. More importantly, we show how the proximity-authentication protocol can be modelled using a one-place buffer (Section 9.4).

9.1. Encoding of the π-calculus

We consider the asynchronous π-calculus [7] given by the following syntax:

$$A, B \ ::= \ \mathbf{0} \mid \bar{x}y.\mathbf{0} \mid x(z).A \mid A|B \mid (\nu z)\ A \mid !A$$

where A and B are process symbols, x denotes a communication channel name, y is a name symbol and z a variable symbol. The output capability $\bar{x}y$ sends the name y over the channel x while the input prefix $x(z).A$ receives the name sent over the channel x and replaces each free occurrence of z in A by that name. This is given by the following reduction rule of the asynchronous π-calculus:

$$\bar{x}y.\mathbf{0} \mid x(z).A \rightarrow_\pi A\{z \leftarrow y\}, \tag{11}$$

where '$\rightarrow_\pi$' denotes the reduction relation of the asynchronous π-calculus.

We let $[\![.]\!]$ denote the encoding function. The output capability $\bar{x}y$ can be modelled in CCA as a child ambient x willing to send the name y to its parent, i.e.

$$[\![\bar{x}y]\!] \mathrel{\widehat{=}} x\big[\uparrow\langle y\rangle.\mathbf{0}\big].$$

The input prefix $x(z).A$ is modelled in CCA as a process willing to receive a value from the child ambient x. However, after the communication has taken place, the ambient x is deleted. That is

$$\big[\!\big[x(z).A\big]\!\big] \mathrel{\widehat{=}} x\!\downarrow\!(z).\texttt{del}\ x.[\![A]\!]$$

The encoding of the remaining constructs of the asynchronous π-calculus is straightforward as follows:

$$\begin{array}{ll}
[\![\mathbf{0}]\!] & \widehat{=}\ \mathbf{0} \\
[\![A \mid B]\!] & \widehat{=}\ [\![A]\!] \mid [\![B]\!] \\
[\![(\nu z)\ A]\!] & \widehat{=}\ (\nu z)\ [\![A]\!] \\
[\![!A]\!] & \widehat{=}\ ![\![A]\!] \\
[\![A\{z \leftarrow y\}]\!] & \widehat{=}\ [\![A]\!]\{z \leftarrow y\}
\end{array}$$

The following theorem establishes the correctness of the encoding.

Theorem 9.1. *If $A \rightarrow_\pi B$ then $[\![A]\!] \rightarrow [\![B]\!]$.*

The proof of this theorem is given in Appendix B.3.

9.2. A one-place buffer

A one-place buffer is a data structure that operates through two actions *push* and *pull*, respectively putting one item in the buffer and taking one item from it. The buffer is full when it contains one item. It is impossible to put one item into a full buffer; it is impossible to take one item from the empty buffer. Let us assume that at the beginning the buffer is empty; the buffer behaves as follows: at first only the *push* operation is possible, after *push* is performed (the buffer contains one item), it is possible to perform *pull*. We model the buffer as an ambient *buf* as follows:

$$buf\left[\begin{array}{l} \uparrow(x).\langle x\rangle.\mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0} \end{array}\right]$$

This ambient waits for its parent to perform *push*, and once a *push* action has been performed the ambient waits for its parent to perform a *pull* action. These actions are specified by the following processes:

$$push(v) \mathrel{\widehat{=}} buf\downarrow\langle v\rangle$$

$$pull(x) \mathrel{\widehat{=}} buf\downarrow(x)$$

For example, suppose the process $push(5).P$ and the process $pull(x).Q$ are running in parallel with the one-place buffer, for some continuation processes P and Q. The first process puts the value 5 in the buffer and continues like the process P while the second takes that value from the buffer and continues like the process $Q\{x \leftarrow 5\}$. This is clearly illustrated by the following derivations with respect to the reduction rules of Table 10:

$$\left.\begin{array}{l}
buf\left[\begin{array}{l} \uparrow(x).\langle x\rangle.\mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0} \end{array}\right] \mid \big(buf\downarrow\langle 5\rangle.P\big) \mid \big(buf\downarrow(x).Q\big) \\
\rightarrow\ \text{(Red Com R3); the buffer receives the value 5} \\
buf\left[\begin{array}{l} \langle 5\rangle.\mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0} \end{array}\right] \mid P \mid \big(buf\downarrow(x).Q\big) \\
\rightarrow\ \text{(Red Com Lc); the buffer is willing to send the value 5} \\
buf\left[\begin{array}{l} \uparrow\langle 5\rangle.\uparrow(x).\langle x\rangle.\mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0} \end{array}\right] \mid P \mid \big(buf\downarrow(x).Q\big) \\
\rightarrow\ \text{(Red Com R2); the buffer sends the value 5 and become empty} \\
buf\left[\begin{array}{l} \uparrow(x).\langle x\rangle.\mathbf{0} \\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0} \end{array}\right] \mid P \mid Q\{x \leftarrow 5\}
\end{array}\right\} \tag{12}$$

9.3. A persistent cell

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. Unlike the one-place buffer discussed above which becomes empty – i.e. looses its content – once a *pull* action is performed, a persistent 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:

$$cell\left[\begin{array}{l} \langle 5\rangle.\mathbf{0} \\ \mid\ !\uparrow().(w).(\langle w\rangle \mid\uparrow\langle w\rangle).\mathbf{0} \\ \mid\ !\uparrow(x).(y).(\langle x\rangle \mid\uparrow\langle\rangle).\mathbf{0} \end{array}\right] \tag{13}$$

A *get* action is performed as followed, where P is a continuation process:

$$cell{\downarrow}\langle\rangle.\mathbf{0} \mid cell{\downarrow}(y).P \tag{14}$$

The component $cell{\downarrow}\langle\rangle.\mathbf{0}$ in Eq. (14) signals to the cell that its parent is willing to perform *get*. The other component receives in the variable y the value sent out by the cell. For example, if Eq. (13) is composed in parallel with Eq. (14) then the value 5 in the cell will be passed to the continuation process P as shown by the following derivations:

$$\left.\begin{array}{l}
cell\left[\begin{array}{l}\quad\langle 5\rangle.\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}\langle\rangle.\mathbf{0} \mid cell{\downarrow}(y).P\\
\to\ (\text{Red Com R3})\\
cell\left[\begin{array}{l}\quad\langle 5\rangle.\mathbf{0}\\ \mid\ (w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}(y).P\\
\to\ (\text{Red Com Lc})\\
cell\left[\begin{array}{l}\mid\ (\langle 5\rangle \mid{\uparrow}\langle 5\rangle).\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}(y).P\\
\to\ (\text{Red Com R2})\\
cell\left[\begin{array}{l}\quad\langle 5\rangle.\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid P\{y \leftarrow 5\}
\end{array}\right\} \tag{15}$$

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

$$cell{\downarrow}\langle 7\rangle.\mathbf{0} \mid cell{\downarrow}().Q \tag{16}$$

The component $cell{\downarrow}\langle 7\rangle.\mathbf{0}$ in Eq. (16) sends to the cell the value 7 to be stored in it. The other component waits until the cell acknowledges that the value has been stored and then continues like Q. This is illustrated by the following derivations:

$$\left.\begin{array}{l}
cell\left[\begin{array}{l}\quad\langle 5\rangle.\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}\langle 7\rangle.\mathbf{0} \mid cell{\downarrow}().Q\\
\to\ (\text{Red Com R3})\\
cell\left[\begin{array}{l}\quad\langle 5\rangle.\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ (y).(\langle 7\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}().Q\\
\to\ (\text{Red Com Lc})\\
cell\left[\begin{array}{l}\mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ (\langle 7\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid cell{\downarrow}().Q\\
\to\ (\text{Red Com R2})\\
cell\left[\begin{array}{l}\quad\langle 7\rangle.\mathbf{0}\\ \mid\ !{\uparrow}().(w).(\langle w\rangle \mid{\uparrow}\langle w\rangle).\mathbf{0}\\ \mid\ !{\uparrow}(x).(y).(\langle x\rangle \mid{\uparrow}\langle\rangle).\mathbf{0}\end{array}\right] \mid Q
\end{array}\right\} \tag{17}$$

Note that at the end of this sequence of derivations, the new value in the cell is 7.

Persistent cells are useful in context-aware applications, e.g. to store user preferences and device profiles. For example, the mode of operation (e.g. *normal* or *silent*) of the smart phone presented in Section 2.1 can be stored on that mobile device using such a data structure. In this case the specification of the smart phone given in Eq. (3) is revised as follows:

$$\mathit{phone}\left[\begin{array}{ll} !\,\mathtt{user\_with}(\mathit{alice})?\mathtt{switchto}\langle \mathit{divert}\rangle.\mathbf{0} \quad \mid & (a)\\ !\,(\mathtt{user\_at}(\mathit{conf}) \wedge \neg\mathtt{user\_with}(\mathit{alice}))?\mathtt{switchto}\langle \mathit{silent}\rangle.\mathbf{0} \quad \mid & (b)\\ !\,(\neg\mathtt{user\_at}(\mathit{conf}) \wedge \neg\mathtt{user\_with}(\mathit{alice}))?\mathtt{switchto}\langle \mathit{normal}\rangle.\mathbf{0} \quad \mid & (c)\\ \\ \mathtt{switchto} \rhd ((z).\mathit{cell}\!\downarrow\langle z\rangle.\mathbf{0} \mid \mathit{cell}\!\downarrow().\mathbf{0}) \quad \mid & (d)\,.\\ \\ \mathit{cell}\left[\begin{array}{l} \langle \mathit{normal}\rangle.\mathbf{0} \quad \mid\\ !\!\uparrow().(w).(\langle w\rangle \mid\uparrow\langle w\rangle).\mathbf{0} \quad \mid\\ !\!\uparrow(x).(y).(\langle x\rangle \mid\uparrow\langle\rangle).\mathbf{0} \end{array}\right] & (e) \end{array}\right] \tag{18}$$

The processes in Eq. (18)-*a*, -*b* and -*c* are the same as in Eq. (3). The process abstraction `switchto` is defined by the process in Eq. (18)-*d* and corresponds to a *put* action – see Eq. (16) – with no continuation process. The process in Eq. (18)-*e* is a persistent cell which stores the phone operation mode; initially the phone is in the mode *normal*, so it rings and vibrates on incoming calls. The mode of the phone changes automatically depending of its user's context, as explained in Section 2.1.

9.4. Health care example revisited

In this section, we reconsider the context-aware hospital bed example and show how a one-place buffer can be used to specify the proximity-authentication protocol. The skeleton of the protocol is given by Eq. (10). The process in Eq. (10)-*a* logs a nurse in when she enters the bed active zone while the process in Eq. (10)-*b* logs that nurse out when she leaves the active zone and then continues like Eq. (10). We model the log-in/log-out mechanism using a one-place buffer (see Section 9.2 for more details) '*buf*' local to the bed as shown in the following specification P_b of the bed ambient. By making it local to the bed ambient, we ensure that the buffer cannot be tampered with (whether accidentally or maliciously) by another ambient. The buffer is specified in Eq. (19)-*b*. The action '$\mathtt{login}\langle \mathit{nurse}\rangle$' *pushes* the nurse's name onto the buffer while the action '$\mathtt{logout}\langle\rangle$' *pulls* that name out of the buffer as specified in Eq. (19)-*c* and Eq. (19)-*d* respectively

$$P_b \;\widehat{=}\; (\nu\, \mathit{buf})\left(\begin{array}{ll} \text{Eq. (10)} & (a)\\ \\ \mid\ \mathit{buf}\left[\begin{array}{l} \uparrow(x).\langle x\rangle.\mathbf{0}\\ \mid\ !(y).\uparrow\langle y\rangle.\uparrow(z).\langle z\rangle.\mathbf{0}\end{array}\right] & (b)\\ \\ \mid\ \mathtt{login} \rhd (v).(\mathit{buf}\downarrow\langle v\rangle.\mathbf{0}) & (c)\\ \mid\ \mathtt{logout} \rhd ().(\mathit{buf}\downarrow(x).\mathbf{0}) & (d)\\ \\ \mid\ !\downarrow(\mathit{req}).\mathbf{0} & (e)\\ \mid\ !::(\mathit{msg}).\mathbf{0} & (f)\\ \mid\ \mathtt{epr} \rhd ().(\mathit{bed}::(x).\mathbf{0} \mid \nu y\ \mathit{bed}::\langle y\rangle.\mathbf{0}) & (g) \end{array}\right) \tag{19}$$

The bed ambient can interact with the patient ambient for example to process its entertainment channel selection; it can also interact with the medicine tray to identify its patient pill container in the tray. These are modelled by Eq. (19)-*e* and Eq. (19)-*f* respectively. Eq. (19)-*g* is an abstraction of the interaction between the nurse ambient and the bed ambient for accessing the patient EPR.

10. Related work

Zimmer [8] proposes a context-awareness calculus which features a hierarchical structure similar to mobile ambients, and a generic multi-agent synchronisation mechanism inspired from the join-calculus. This work has been extended in [9] by Bucur and Nielson to enable ambients to publish context information upwards in the hierarchy of ambients. In both work, a piece of primary context information is a capability modelled by a named macro, similar to the notion of process call in CCA. Unlike their approach and to preserve the autonomy of ambients, we do not allow an ambient to remotely create a process abstraction in another ambient. Moreover, none of these process calculi can handle context-guarded capabilities.

Roman et al. [14] propose *Context UNITY* as a formal model for expressing aspects of context-aware computations. In this model, context is provided through *exposed variables* and existential quantification is used for context discovery. A verification mechanism is provided based on the underlying UNITY proof rules. Lopes and Fiadeiro [15] propose the use of abstract data types in CommUnity for modelling context explicitly according to application domains. They use four special observables, similar to exposed variables found in [14], to provide context information on the entire system. In CCA context information is distributed among system components. Bouzeghoub et al. [16] propose a system to perform situation-aware adaptive recommendation of information to assist mobile users in a campus environment. They use, as many other work [17–19,16,20], OWL ontology language to model context.

Milner [21–23] introduced bigraphical reactive systems (BRSs) as a unifying framework for designing models of concurrent and mobile systems. These reactive systems are defined as a set of rewriting rules together with an initial bigraph on which the rules operate. A bigraph is a particular kind of graphs which allows for the representation of communication among nodes as well as their spatial configuration (nodes may be nested within each other). In [24], Birkedal et al. attempt the modelling of context-aware 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 our model context-guarded capabilities and process calls – which are primitives constructs in CCA – are used to acquire context information.

In [25–28] event-condition-action (ECA in short) rules are used to model the behaviours of context-aware applications. An ECA rule has the general form:

$$\texttt{ON}\ \mathit{event}\ \texttt{IF}\ \mathit{condition}\ \texttt{DO}\ \mathit{action},$$

where *event* signals a change in the external environment, *condition* is a Boolean expression about the state of the system and *action* is a computation. The meaning of this rule is: when the *event* occurs, if the *condition* is true then the *action* is performed. In CCA, assuming that the system is modelled as an ambient, an ECA rule can be represented as follows:

$$!\,(\mathit{event} \wedge \mathit{condition})?\mathit{action},$$

where

- *event* is a context expression about the external context of that ambient;
- *condition* is a context expression about the internal context of that ambient; and
- *action* is a process.

The replication operator '!' means that the rule is applied forever. Samples of such a rule are given, e.g., in Eq. (3).

11. Conclusion and future work

In this paper we have presented CCA, a calculus of mobile systems that are context-aware. This calculus inherits the mobility model of MA and introduces constructs for modelling context-awareness. The first construct for modelling context-awareness is a *context-guarded capability* whereby a capability is guarded by a context expression which states properties that the environment must meet for the capability to be performed. Computation so becomes context-dependent or context-aware. A formal model of context has been defined for representing the environment of a CCA process, and a spatial logic for expressing properties of contexts has been proposed. A formula in this logic is called *a context expression*. Common context properties such as location, nearby people and resources, and social situations can be expressed in a natural manner as shown by the many examples given in this paper. As customary, the notions of satisfaction and validity have been formally defined for context expressions, based on our formal model of context.

In addition to the notion of *context-guarded capability*, CCA enables the use of *process abstractions* as a mean to provide context information. Each ambient can redefined a process abstraction to suit its own needs and capacities. So a visiting mobile ambient making a process call to that process abstraction will execute the definition available at its current location. Therefore, a process call might behave differently when performed at different locations.

The syntax, formal semantics and expressiveness of CCA have been presented in this paper. We have shown that CCA encodes the π-calculus. We propose a new equivalence theory of processes which allows to tell whether two context-aware processes behave the same. We prove that this relation is a congruence. Finally, many motivating examples have been presented including a real-world case study of a context-aware hospital bed developed in the *Hospital of the Future* project at the Centre for Pervasive Health Care, Denmark [11,12].

In future work, we will investigate techniques (such as type systems) for verifying key properties of context-aware applications such as safety, security and privacy. As customary in process calculi, type-checking algorithms can be constructed to verify statically or dynamically the properties of processes.

Publish/subscribe is an asynchronous messaging paradigm where senders (publishers) send messages only to the applications that are interested in receiving the messages without knowing the identities of the receivers (subscribers). In many publish/subscribe systems, publishers post messages to an intermediary message broker and subscribers register subscriptions with that broker which then forwards the messages to them as illustrated in Fig. 2. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology. In future work, we will model and analyse publish/subscribe systems using our calculus. Here, we discuss how the simplified publish/subscribe system depicted in Fig. 2 can be modelled in CCA. The publisher, the broker and the subscribers can be modelled as ambients named *publisher*, *broker*, *subscriber1*, *subscriber2*, *subscriber3*, respectively:

- The publisher creates new messages and sends them to the broker:

$$\mathit{publisher}\big[!\,(\nu\ \mathit{msg})\ \mathit{broker} :: \langle \mathit{msg}\rangle.\mathbf{0}\big] \tag{20}$$

- The broker receives the messages and forwards them to the subscribers:

$$\mathit{broker}\left[\,!\,\mathit{publisher} :: (x).\left(\begin{array}{l}\mathit{subscriber1} :: \langle x\rangle.\mathbf{0}\ |\\ \mathit{subscriber2} :: \langle x\rangle.\mathbf{0}\ |\\ \mathit{subscriber3} :: \langle x\rangle.\mathbf{0}\end{array}\right)\right] \tag{21}$$

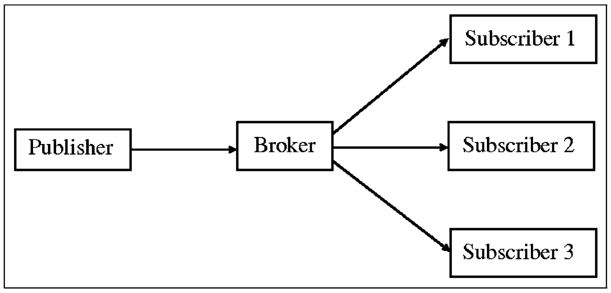

Fig. 2. Publish/subscribe system.

Table 11
Free names in processes.

$\mathbf{fn}(\mathbf{0})$	$\widehat{=}$	$\emptyset$	$\mathbf{fn}(\alpha\ x\langle\tilde{y}\rangle.P)$	$\widehat{=}$	$\mathbf{fn}(\alpha) \cup \{x\} \cup \tilde{y} \cup \mathbf{fn}(P)$
$\mathbf{fn}(P \mid Q)$	$\widehat{=}$	$\mathbf{fn}(P) \cup \mathbf{fn}(Q)$	$\mathbf{fn}(\alpha\ (\tilde{y}).P)$	$\widehat{=}$	$\mathbf{fn}(\alpha) \cup (\mathbf{fn}(P) \backslash \tilde{y})$
$\mathbf{fn}((\nu n)\ P)$	$\widehat{=}$	$\mathbf{fn}(P) \backslash \{n\}$	$\mathbf{fn}(\alpha\ \langle\tilde{y}\rangle.P)$	$\widehat{=}$	$\mathbf{fn}(\alpha) \cup \tilde{y} \cup \mathbf{fn}(P)$
$\mathbf{fn}(n[P])$	$\widehat{=}$	$\mathbf{fn}(P) \cup \{n\}$	$\mathbf{fn}(\uparrow)$	$\widehat{=}$	$\emptyset$
$\mathbf{fn}(!P)$	$\widehat{=}$	$\mathbf{fn}(P)$	$\mathbf{fn}(n\uparrow)$	$\widehat{=}$	$\{n\}$
$\mathbf{fn}(\kappa ? P)$	$\widehat{=}$	$\mathbf{fn}(\kappa) \cup \mathbf{fn}(P)$	$\mathbf{fn}(\downarrow)$	$\widehat{=}$	$\emptyset$
$\mathbf{fn}(x \rhd (\tilde{y}).P)$	$\widehat{=}$	$\{x\}$	$\mathbf{fn}(n\downarrow)$	$\widehat{=}$	$\{n\}$
$\mathbf{fn}(\mathtt{in}\ n)$	$\widehat{=}$	$\{n\}$	$\mathbf{fn}(::)$	$\widehat{=}$	$\emptyset$
$\mathbf{fn}(\mathtt{out})$	$\widehat{=}$	$\emptyset$	$\mathbf{fn}(n ::)$	$\widehat{=}$	$\{n\}$
			$\mathbf{fn}(\epsilon)$	$\widehat{=}$	$\emptyset$

Table 12
Free names in context expressions.

$\mathbf{fn}(\mathbf{True})$	$\widehat{=}$	$\emptyset$	$\mathbf{fn}(\kappa_1 \wedge \kappa_2)$	$\widehat{=}$	$\mathbf{fn}(\kappa_1) \cup \mathbf{fn}(\kappa_2)$
$\mathbf{fn}(n = m)$	$\widehat{=}$	$\{n, m\}$	$\mathbf{fn}(\mathtt{new}(n, \kappa))$	$\widehat{=}$	$\mathbf{fn}(\kappa) \backslash \{n\}$
$\mathbf{fn}(\bullet)$	$\widehat{=}$	$\emptyset$	$\mathbf{fn}(\oplus\kappa)$	$\widehat{=}$	$\mathbf{fn}(\kappa)$
$\mathbf{fn}(\neg\kappa)$	$\widehat{=}$	$\mathbf{fn}(\kappa)$	$\mathbf{fn}(\diamond\kappa)$	$\widehat{=}$	$\mathbf{fn}(\kappa)$
$\mathbf{fn}(\kappa_1 \mid \kappa_2)$	$\widehat{=}$	$\mathbf{fn}(\kappa_1) \cup \mathbf{fn}(\kappa_2)$	$\mathbf{fn}(\exists x.\kappa)$	$\widehat{=}$	$\mathbf{fn}(\kappa) \backslash \{x\}$
$\mathbf{fn}(n[\kappa])$	$\widehat{=}$	$\mathbf{fn}(\kappa) \cup \{n\}$			

- Each subscriber $subscriber_i$ receives the messages from the broker and performs some continuation process P_i, $1 \leqslant i \leqslant 3$:

$$subscriber1\big[!\ broker :: (y).P_1\big] \mid subscriber2\big[!\ broker :: (y).P_2\big] \mid subscriber3\big[!\ broker :: (y).P_3\big] \tag{22}$$

So the whole system is modelled by the following process:

Eq. (20) | Eq. (21) | Eq. (22).

Acknowledgments

The authors would like to thank the anonymous referees for their critical review of this paper.

Appendix A. Free names

The set of free names in a process is defined inductively as in Table 11. The free names of a context expression are calculated as in Table 12.

Appendix B. Proofs

In this appendix, we give the proofs of all the theorems that have not been proved in the main body of the paper.

B.1. Proof of Theorem 5.1

Proof of Theorem 5.1. The proof of this theorem is given by induction on ℓ, the number of connectives in the formula κ.

Base case We prove that the theorem holds for $\ell = 0$, i.e. the formula κ is of one of the following forms: **True**, $n = n$ and $\bullet$, for some name n.

- For **True**: the proof is obvious because all context satisfies **True**.
- For $n = n$: the proof is also obvious because all context satisfies $n = n$.

- For $\bullet$: the proof is straightforwards because a context satisfies $\bullet$ if and only if that context is equal to the hole context $\odot$. So $E \models \bullet$ implies $E = \odot$; and $E = \odot = E'$ implies $E' \models \bullet$.

Induction step We suppose that the theorem holds for all formula κ having up to ℓ connectives and prove that it also holds for all formula having $\ell + 1$ connectives. We distinguish the following cases corresponding to the syntactic categories of formulae:

Negation Suppose $E \models \neg\kappa$ and $E = E'$ and $E' \not\models \neg\kappa$. This implies that $E = E'$ and $E' \models \kappa$. Using the induction hypothesis it follows that $E \models \kappa$, which is a contradiction. So we conclude that $E' \models \neg\kappa$.
Location By definition, $E \models n[\kappa]$ if and only if $E = n[E'']$ for some context E'' such that $E'' \models \kappa$. So $E = E'$ implies $E' = n[E'']$. It follows from $E' = n[E'']$ and $E'' \models \kappa$ that $E' \models n[\kappa]$ holds.
Revelation The proof is similar to that of Location.
Parallel The proof is similar to that of Location.
Exists The proof is similar to that of Location.
Next The proof is trivial from Theorem 4.1.
Somewhere The proof is trivial from Theorem 4.1.
Conjunction Suppose $E = E'$ and $E \models \kappa_1 \wedge \kappa_2$, where $\kappa_1 \wedge \kappa_2$ contains $\ell + 1$ connectives. It follows that κ_1 and κ_2 contains each at most ℓ connectives. By definition $E \models \kappa_1 \wedge \kappa_2$ if an only if $E \models \kappa_1$ and $E \models \kappa_2$. Using the induction hypothesis, it follows that $E' \models \kappa_1$ and $E' \models \kappa_2$. We then conclude that $E' \models \kappa_1 \wedge \kappa_2$. □

B.2. Proof of Theorem 8.1

Proof of Theorem 8.1. We first recall the definition of a congruence. A relation $\mathcal{R}$ is a congruence if it has the following properties, for all processes P, Q and R:

- Reflexivity: $P \,\mathcal{R}\, P$
- Symmetry: if $P \,\mathcal{R}\, Q$ then $Q \,\mathcal{R}\, P$
- Transitivity: if $P \,\mathcal{R}\, Q$ and $Q \,\mathcal{R}\, R$ then $P \,\mathcal{R}\, R$
- Precongruence: if $P \,\mathcal{R}\, Q$ then $C(P) \,\mathcal{R}\, C(Q)$, for all generalised context C.

The proof that $\simeq$ is reflexive, symmetric and transitive is straightforward from the properties of the predicate connective $\Leftrightarrow$ as used in Definition 8.4. Now we assume that $P \simeq Q$ and prove that $C(P) \simeq C(Q)$, for some processes P and Q, and some generalised context C. Note that if C_1 and C_2 are generalised contexts, then so is $C_1(C_2)$ which is obtained by filling each hole in C_1 with a copy of C_2.

Let C' be a generalised context such that $C'(C(P)) \perp\!\!\perp_n^\kappa$ is true, for some context expression κ and ambient name n. We prove that $C'(C(Q)) \perp\!\!\perp_n^\kappa$ is also true as follows:

$$\begin{aligned}
& C'\big(C(P)\big) \perp\!\!\perp_n^\kappa \\
\Leftrightarrow\ & \big\{C'\big(C(P)\big) \equiv \big(C'(C)\big)(P) \text{ and Definition 8.3}\big\} \\
& \big(C'(C)\big)(P) \perp\!\!\perp_n^\kappa \\
\Leftrightarrow\ & \{P \simeq Q\} \\
& \big(C'(C)\big)(Q) \perp\!\!\perp_n^\kappa \\
\Leftrightarrow\ & \big\{C'\big(C(P)\big) \equiv \big(C'(C)\big)(P) \text{ and Definition 8.3}\big\} \\
& C'\big(C(Q)\big) \perp\!\!\perp_n^\kappa
\end{aligned}$$

We then deduce that $C(P) \simeq C(Q)$. □

B.3. Proof of Theorem 9.1

Proof of Theorem 9.1. The main reduction rule of the π-calculus is defined in Eq. (11). This rule captures the ability of processes to communicate through channels. The proof is as follows:

$$\begin{aligned}
& [\![\bar{x}y.\mathbf{0} \mid x(z).A]\!] \\
\widehat{=}\ & \big\{\text{from the definition of } [\![.]\!]\big\} \\
& [\![\bar{x}y.\mathbf{0}]\!] \mid [\![x(z).A]\!] \\
\widehat{=}\ & \big\{\text{from the definition of } [\![.]\!]\big\} \\
& x\big[\uparrow\langle y\rangle.\mathbf{0}\big] \mid x\!\downarrow\!(z).\texttt{del}\ x.[\![A]\!] \\
\rightarrow\ & \big\{\text{from (Red Com R1) in Table 10}\big\} \\
& x[\mathbf{0}] \mid \big(\texttt{del}\ x.[\![A]\!]\big)\{z \leftarrow y\}
\end{aligned}$$

$$\begin{aligned}
\equiv &\quad \big\{\text{from (S12) in Table 9}\big\}\\
&x[\mathbf{0}] \mid \mathtt{del}\ x.\big([\![A]\!]\{z \leftarrow y\}\big)\\
\equiv &\quad \big\{\text{from (S22) in Table 9}\big\}\\
&[\![A]\!]\{z \leftarrow y\}\\
\widehat{=} &\quad \big\{\text{from the definition of } [\![.]\!]\big\}\\
&\big[\!\!\big[A\{z \leftarrow y\}\big]\!\!\big]
\end{aligned}$$

So, we conclude that

$$\big[\!\!\big[\bar{x}y.\mathbf{0} \mid x(z).A\big]\!\!\big] \to \big[\!\!\big[A\{z \leftarrow y\}\big]\!\!\big]. \qquad \square$$

References

[1] M. Weiser, Some computer science issues in ubiquitous computing, Communications of the ACM 36 (7) (1993) 75–84.
[2] A.K. Dey, G.D. Abowd, Towards a better understanding of context and context-awareness, in: Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, ACM Press, New York, 2000.
[3] L. Cardelli, A. Gordon, Mobile ambients, Theoret. Comput. Sci. 240 (2000) 177–213.
[4] J.H. Morris, Lambda-calculus models of programming languages, PhD thesis, MIT, 1968.
[5] R. Milner, Functions as processes, Math. Structures Comput. Sci. 2 (2) (1992) 119–141.
[6] R. Milner, Communication and Mobile Systems: The π-Calculus, Cambridge University Press, 1999.
[7] D. Sangiorgi, D. Walker, The π-Calculus: A Theory of Mobile Processes, Cambridge University Press, 2001.
[8] P. Zimmer, A calculus for context-awareness, Tech. Rep., BRICS, 2005.
[9] D. Bucur, M. Nielsen, Secure data flow in a calculus for context awareness, in: Concurrency, Graphs and Models, in: Lecture Notes in Comput. Sci., vol. 5065, Springer, 2008, pp. 439–456.
[10] M. Bugliesi, G. Castagna, S. Crafa, Access control for mobile agents: The calculus of boxed ambients, ACM Trans. Program. Languages Systems 26 (1) (2004) 57–124.
[11] J. Bardram, Hospitals of the future-ubiquitous computing support for medical work, Hospitals Workshop Ubihealth 2003, 2003.
[12] J. Bardram, Applications of context-aware computing in hospital work-examples and design principles, in: Proceedings of ACM Symposium on Applied Computing, ACM Press, 2004, pp. 1574–1579.
[13] R. De Nicola, M.C.B. Hennessy, Testing equivalences for processes, Theoret. Comput. Sci. 34 (1984) 83–133.
[14] G.-C. Roman, C. Julien, J. Payton, Modeling adaptive behaviors in context UNITY, in: Fundamental Aspects of Software Engineering, Theoret. Comput. Sci. 376 (3) (2007) 185–204.
[15] A. Lopes, J.L. Fiadeiro, Algebraic semantics of design abstractions for context-awareness, in: Recent Trends in Algebraic Development Techniques, in: Lecture Notes in Comput. Sci., vol. 3423, Springer, 2005, pp. 79–93.
[16] A. Bouzeghoub, K.N. Do, L.K. Wives, Situation-aware adaptive recommendation to assist mobile users in a campus environments, in: International Conference on Advanced Information Networking and Applications, IEEE Computer Society, 2009, pp. 503–508.
[17] H. Chen, T. Finin, A. Joshi, An ontology for context-aware pervasive computing environments, Knowl. Eng. Rev. 18 (3) (2003) 197–207.
[18] X.H. Wang, D.Q. Zhang, T. Gu, H.K. Pung, Ontology based context modeling and reasoning using owl, in: Pervasive Computing and Communications Workshops, IEEE International Conference on, 2004, p. 18.
[19] R.M. Pessoa, C.Z. Calvi, J.P. Filho, C.G. de Farias, R. Neisse, Semantic context reasoning using ontology based models, in: A. Pras, M. van Sinderen (Eds.), Dependable and Adaptable Networks and Services, 13th Open European Summer School and IFIP TC6.6 Workshop (EUNICE), in: Lecture Notes in Comput. Sci., vol. 4606, Springer-Verlag, Germany, 2007, pp. 44–51.
[20] G. Hynes, V. Reynolds, M. Hauswirth, Enabling mobility between context-aware smart spaces, in: International Conference on Advanced Information Networking and Applications, IEEE Computer Society, 2009, pp. 255–260.
[21] J. Krivine, R. Milner, A. Troina, Stochastic bigraphs, in: Proc. of MFPS'08, 24th Conference on the Mathematical Foundations of Programming Semantics, in: Electron. Notes Theor. Comput. Sci., vol. 218, Elsevier, 2008, pp. 73–96.
[22] R. Milner, Pure bigraphs: structure and dynamics, Inform. and Comput. 204 (2006) 60–122.
[23] O.H. Jensen, R. Milner, Bigraphs and Mobile Processes (Revised), Tech. Rep. UCAM-CL-TR-580, University of Cambridge, 2004.
[24] L. Birkedal, S. Debois, E. Elsborg, T. Hildebr, H. Niss, Bigraphical models of context-aware systems, in: IT University of Copenhagen, Springer-Verlag, 2006, pp. 187–201.
[25] E.E. Almeida, J.E. Luntz, D.M. Tilbury, Event-condition-action systems for reconfigurable logic control, IEEE Trans. Knowl. Data Eng. 4 (2) (2007) 167–181.
[26] A. Moon, H. Kim, H. Kim, S. Lee, Context-aware active services in ubiquitous computing environments, ETRI Journal 29 (2) (2007) 169–178.
[27] T. Beer, J. Rasinger, W. Höpken, M. Fuchs, H. Werthner, Exploiting E-C-A rules for defining and processing context-aware push messages, Lecture Notes in Comput. Sci. 4824/2007 (2007) 199–206.
[28] J. Bae, H. Bae, S.-H. Kang, Y. Kim, Automatic control of workflow processes using ECA rules, IEEE Trans. Knowl. Data Eng. 16 (8) (2004) 1010–1023.