Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 98 – 105

Procedia Computer Science

6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

A Privacy Type System for Context-aware Mobile Ambients

François Siewe

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

Abstract

Thanks to the advances in technologies, ubiquitous computing (ubicomp) is developing fast with the proliferation of smart devices such as smart phones and tablet computers. However, privacy is an important concern in ubicomp; unless users are confident enough that their privacy is protected, many will be deterred from using such systems. This paper proposes a privacy type system that controls the behaviour of concurrent, context-aware and mobile processes to ensure that private information are not accidentally disclosed. We prove the subject reduction property, which guarantees that a well-typed process is safe and cannot disclose private information to an unauthorised party.

Peer-review under responsibility of the Conference Program Chairs

Keywords:

privacy, type system, ubiquitous computing, security, CCA

1. Introduction

Thanks to the advances in technologies, the vision of ubiquitous computing (ubicomp)[1] is increasingly becoming a reality with the proliferation of smart handheld devices such as smart phones and tablet computers capable of providing the user with relevant information and services anytime and anywhere. These smart devices are equipped with a variety of sensors to collect information upon the user context and share these information via a network to enable timely adaptation to changes in the user context. These might be private or highly sensitive personal information such as the user's location, activity, or personal health information that must be protected from falling into the wrong hands. Unless a mechanism is put in place that gives users enough confidence that their privacy is protected, many will be deterred from using such systems. However, privacy is a subjective concept based on personal perceptions of risk and benefit[2]; this makes it more difficult to enforce compared to traditional access control. In general, people are likely to disclose personal information in exchange of services if they believe the benefit outweighs the potential cost of these information being misused[2,3].

The Calculus of Context-aware Ambients (CCA)[4] is a process calculus for modelling ubicomp systems. The main features of the calculus include *context-awareness*, *mobility* and *concurrency*. The concept of *ambient*, inherited

* Corresponding author. Tel.: +44-116-257-7938 ; fax: +44-116-257-7936.
E-mail address: fsiewe@dmu.ac.uk

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.034

Table 1. Syntax of CCA

P, Q	::=	$\mathbf{0} \mid \text{`}P \vert Q\text{'} \mid (\nu\, n : W)\, P \mid (\nu\, g : \mathtt{gr})\, P \mid !P \mid n[P] \mid \kappa?M.P$
M	::=	$\mathtt{in}\, n \mid \mathtt{out} \mid \alpha\, \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}) \mid \alpha\, \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\})$
α	::=	$\uparrow \mid n \uparrow \mid \downarrow \mid n \downarrow \mid :: \mid n :: \mid \epsilon$
κ	::=	$\mathtt{true} \mid \bullet \mid n = m \mid n[\kappa] \mid \neg\kappa \mid \kappa_1 \vert \kappa_2 \mid \kappa_1 \wedge \kappa_2 \mid \oplus\kappa \mid \diamond\kappa$

from[5], represents an abstraction of a place where computation may happen. An ambient may contain other ambients called child ambients organised into a tree structure. Such a hierarchy can be used to model any entity in a ubicomp system –whether physical, logical, mobile or immobile– as well as the environment (or context) of that entity[4]. In CCA like in a ubicomp system, an ambient can exchange messages, can be mobile and can be aware of the presence of other ambients. These interactions among ambients, if not properly regulated, may lead to unwanted disclosure of private information, whether directly or indirectly.

This paper proposes a novel type system that constrains the behaviour of CCA processes to ensure that private information are not accidentally disclosed. This type system enables an ambient to specify who can read its context information, who can share that information with a third party, and who that third party might be. Type checking guarantees that a well-typed process cannot violate the privacy of any ambient. The main contributions of this work are threefold:

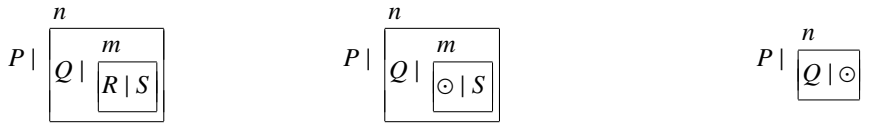
- We propose a syntax for types (Sect. 3.1); its innovative features include notations for describing privacy types. Mobility types and exchange types can also be specified. The privacy type of an ambient specifies the groups of the ambients that can sense that ambient context information, the groups of the ambients that can share that information with a third party, and the groups that third party must belong to. In this way, *primary* and *secondary* use of context information can be controlled. Type annotations are added to the syntax of CCA in Sect. 2.1.
- We formalise the proposed type system using typing rules (Sect. 3.2); only processes that can be typed using these rules are *well-typed*. These rules are used to check statically (i.e. at compile time) the well-typedness of processes.
- We prove, as customary in process calculi, the subject reduction property of the proposed type system (Sect. 4). This property states that a well-typed process can only reduce to well-typed processes; therefore it is guaranteed that run-time errors (e.g. exchange of a message of wrong type) and violation of privacy cannot occur during the execution of well-typed processes.

2. Syntax and Semantics of Typed CCA

The syntax of typed CCA is depicted in Table 1, featuring three syntactic categories: processes (denoted by P or Q), capabilities (denoted by M) and context-expressions (denoted by κ). We assume a countably-infinite set of names, elements of which are written in lower-case letters, e.g. n, g, x and y. Each name has a type, either a message type W (defined later in Sect. 3.1) or the group type gr.

2.1. Syntax

The process $\mathbf{0}$, aka *inactivity process*, does nothing and terminates immediately. The process $P|Q$ denotes the concurrent execution of the processes P and Q. The process $(\nu\, n : W)\, P$ creates a new name n of type W and the scope of that name is limited to the process P. The replication $!P$ denotes a process which can always create a new copy of P, i.e. $!P$ is equivalent to $P|!P$. Replication, first introduced in the π-Calculus[6], can be used to implement both iteration and recursion. The process $n[P]$ denotes an ambient named n whose behaviours are described by the process P. A context expression κ specifies a property upon the surrounding environment (aka context) of a process. A *context-guarded prefix* $\kappa?M.P$ is a process that waits until the environment satisfies the context expression κ, then performs the capability M and continues like the process P. The dot symbol '.' denotes the sequential composition of processes. We let $M.P$ denote the process $\mathtt{true}?M.P$, where true is a context expression satisfied by all context.



(a) System model (b) Context of process R (c) Context of ambient m

Fig. 1. Graphical illustration of the context of a process

Table 2. Satisfaction relation for context expressions

$C \models \texttt{true}$.	$C \models n = n$.	$C \models \bullet$ iff $C = \odot$.	$C \models \neg\kappa$ iff $C \not\models \kappa$.	$C \models \kappa_1 \wedge \kappa_2$ iff $C \models \kappa_1$ and $C \models \kappa_2$.
$C \models \kappa_1 \mid \kappa_2$ iff exist C_1, C_2 such that $C = C_1 \mid C_2$ and $C_1 \models \kappa_1$ and $C_2 \models \kappa_2$.		$C \models n[\kappa]$ iff exists C' such that $C = n[C']$ and $C' \models \kappa$.		
$C \models \oplus\kappa$ iff exist C', n such that $C = n[C']$ and $C' \models \kappa$.		$C \models \Diamond\kappa$ iff $C \models \kappa$ or exist C', n such that $C = n[C']$ and $C' \models \Diamond\kappa$.		

Ambients exchange messages using the output capability α $\texttt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\})$ to send a record containing the names $z_1, \ldots, z_\ell$ to a location α, and the input capability α $\texttt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\})$ to receive a record containing names of specified types from a location α, where $l_1, \ldots, l_\ell$ are the fields of this record type (see Table 4). 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 executing ambient itself. The mobility capability `in` moves the executing ambient into a sibling ambient, while the capability `out` moves the ambient that performs it out of that ambient's parent.

In CCA, a context is modelled as a process with a hole in it[4]. The hole (denoted by $\odot$) in a context represents the position of the process that context is the context of. For example, suppose a system is modelled by the process $P \mid n[Q \mid m[R \mid S]]$. So, the context of the process R in that system is $P \mid n[Q \mid m[\odot \mid S]]$, and that of the ambient named m is $P \mid n[Q \mid \odot]$ as depicted graphically in Fig. 1. Thus the context of a typed CCA process can be described by the grammar: $C ::= \mathbf{0} \mid \odot \mid n[C] \mid$ '$C|P$' $\mid (\nu\ n : W)\ C \mid (\nu\ g : \texttt{gr})\ C$. A property of a context can be described by a formula κ (see Table 1) called a *context expression* (CE in short). The CE `true` always holds. A CE $n = m$ holds if the names n and m are lexically identical. The CE $\bullet$ holds solely for the hole context, i.e. the position of the process evaluating that context expression. Propositional operators such as negation ($\neg$) and conjunction ($\wedge$) expand their usual semantics to context expressions. A CE $\kappa_1|\kappa_2$ holds for a context if that context is a parallel composition of two contexts such that κ_1 holds for one and κ_2 holds for the other. A CE $n[\kappa]$ holds for a context if that context is an ambient named n such that κ holds inside that ambient. A CE $\oplus\kappa$ holds for a context if that context has a child context for which κ holds. A CE $\Diamond\kappa$ holds for a context if there exists somewhere in that context a sub-context for which κ holds. The operator $\Diamond$ is called *somewhere modality*, while $\oplus$ is aka *spatial next modality*. Common context properties such as the location of the user, with whom the user is and what resources are nearby can be expressed using the following predicates:

- $\texttt{has}(n) = \Diamond(\bullet \mid n[\texttt{true}] \mid \texttt{true})$: holds if the executing ambient contains an ambient named n
- $\texttt{at}(n) = \Diamond n[\oplus(\bullet \mid \texttt{true})] \mid \texttt{true}$: holds if the executing ambient is located at an ambient named n
- $\texttt{with}(n) = \Diamond(n[\texttt{true}] \mid \oplus(\bullet \mid \texttt{true}))$: holds if the executing ambient is with an ambient named n.
- $\texttt{at2}(n, m) = n[m[\texttt{true}] \mid \texttt{true}] \mid \texttt{true}$: holds if ambient m is located at a top ambient named n.
- $\texttt{with2}(n, m) = n[\texttt{true}] \mid m[\texttt{true}] \mid \texttt{true}$: holds if ambient m is (co-located) with a top ambient named n.

2.2. Semantics

The formal semantics of CEs is given in Table 2, where the notation '$C \models \kappa$' means that the context C satisfies the context expression κ. The operational semantics of typed 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 3, where $\texttt{fn}(X)$ (resp. $\texttt{fg}(X)$) denotes the set of all the names (resp. groups) that occur free in the term X. These axioms allow the manipulation of the structure of processes. It follows from these axioms that the structural congruence is a commutative monoid for $(\mathbf{0}, |)$. These axioms are inherited from CCA[4], differ just with the type annotations. For instance, the axiom (S22) says that a capability guarded with `true` is the same as that capability

Table 3. Structural congruence for processes

(S1) $P \equiv P$ **(S2)** $P \equiv Q \Rightarrow Q \equiv P$ **(S3)** $P \equiv Q, Q \equiv R \Rightarrow P \equiv R$ **(S4)** $P \equiv Q \Rightarrow (\nu n : W)\, P \equiv (\nu n : W)\, Q$ **(S5)** $P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$
(S6) $P \equiv Q \Rightarrow\, !P \equiv !Q$ **(S7)** $P \equiv Q \Rightarrow n[P] \equiv n[Q]$ **(S8)** $P \mid Q \equiv Q \mid P$ **(S9)** $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$
(S10) $(\nu n : W_1)\,(\nu m : W_2)\, P \equiv (\nu m : W_2)\,(\nu n : W_1)\, P$ **(S11)** $(\nu n : W)\, P \mid Q \equiv (\nu n : W)\,(P \mid Q)$ if $n \notin \mathbf{fn}(Q)$
(S12) $(\nu n : W)\, m[P] \equiv m[(\nu n : W)\, P]$ if $n \neq m$ **(S13)** $(\nu g_1 : \mathtt{gr})\,(\nu g_2 : \mathtt{gr})\, P \equiv (\nu g_2 : \mathtt{gr})\,(\nu g_1 : \mathtt{gr})\, P$ **(S22)** $\mathtt{true}?M.P \equiv M.P$
(S14) $(\nu g : \mathtt{gr})\,(\nu n : W)\, P \equiv (\nu n : W)\,(\nu g : \mathtt{gr})\, P$ if $g \notin \mathbf{fg}(W)$ **(S15)** $(\nu g : \mathtt{gr})\, P \mid Q \equiv (\nu g : \mathtt{gr})\,(P \mid Q)$ if $g \notin \mathbf{fg}(Q)$
(S16) $(\nu g : \mathtt{gr})\, m[P] \equiv m[(\nu g : \mathtt{gr})\, P]$ **(S17)** $!P \equiv P \mid !P$ **(S18)** $P \mid \mathbf{0} \equiv P$ **(S19)** $!\mathbf{0} \equiv \mathbf{0}$ **(S20)** $(\nu n : W)\, \mathbf{0} \equiv \mathbf{0}$ **(S21)** $(\nu g : \mathtt{gr})\, \mathbf{0} \equiv \mathbf{0}$

Table 4. Types

W	::=	$\mathtt{amb}(g)[X, T, U] \mid Z \mid \mu Z.W$	**Message types**
T	::=	$\mathtt{shh} \mid \{l_1 : W_1, \cdots, l_\ell : W_\ell\}$	**Exchange types**
X	::=	$\curvearrowright \mid \not\curvearrowright$	**Mobility types**
U	::=	$\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$	**Privacy types**

without guards. Similarly, the relation '$\rightarrow$' is the same as for CCA[4], except for the type annotations. For example the rule Eq. 1 of typed CCA is so obtained from the reduction rule Eq. 2 of CCA, where $P\{z_1/y_1, \ldots, z_\ell/y_\ell\}$ denotes the process obtained by replacing each free occurrence of the name y_i in P by z_i, $i = 1, \ldots, \ell$.

$$\mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q \quad \rightarrow \quad P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q \tag{1}$$

$$\mathtt{recv}(\{l_1 = y_1, \ldots, l_\ell = y_\ell\}).P \mid \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q \quad \rightarrow \quad P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q \tag{2}$$

3. A Privacy Type System for CCA

3.1. Types

The types in Table 4 are devised to regulate communication, mobility and context-awareness in a manner that protects the privacy of individual ambients. There are four main categories of types: message types (denoted by W), exchange types (T), mobility types (X) and privacy types (U). A message type $\mathtt{amb}(g)[X, T, U]$ is the type of an ambient of group g, mobility type X, exchange type T and privacy type U. To enable an ambient to exchange names of that ambient type, we use a *recursive type* of the form $\mu Z.W$, where Z is a type variable possibly occurring free in W. Note that all recursive types that can be defined in our type system are guarded, either by the keyword amb. We write $W \sim W'$ to mean that two message types W and W' are equal. Similarly to the type system for mobile ambients[7], the mobility types state whether a process is mobile ($\curvearrowright$) or immobile ($\not\curvearrowright$). The exchange types specify the communication interface of a process: shh if the process has no communication capability; and $\{l_1 : W_1, \ldots, l_\ell : W_\ell\}$ if the process has the capability of communicating a value of this record type, where $l_i, 1 \leq i \leq \ell$ are the fields of the record type and are all different. By convention, the type of an empty record (i.e. $\ell = 0$) is denoted by $\mathbf{1}$. A subtyping relation '$\leq$' is defined over exchange types as follows, where n and m are non-negative integers:

$$\{l_1 : W_1, \ldots, l_{n+m} : W_{n+m}\} \leq \{l_1 : W_1, \ldots, l_n : W_n\} \leq \mathbf{1} \leq \mathtt{shh}. \tag{3}$$

The privacy types control the information flow among processes to ensure that only authorised ambients can access or share a specific piece of context information. A privacy type has the form $\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$, where $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{I}$ are sets of groups such that:

- $\mathcal{R}$: only ambients of these groups are allowed to *read* the context of any ambient of this privacy type. We say that an ambient n reads the context of another ambient m if the name m occurs freely in a context-expression κ or in a location term α (see grammar in Table 1) in the body of the ambient n. The notation $\mathcal{R}$ comes from the action *read*.
- $\mathcal{O}$: only ambients of these groups are allowed to *output to* (or share with) other ambients the context of any ambient of this privacy type. The notation $\mathcal{O}$ comes from the action *output*.

Table 5. Type judgements

$\Gamma \vdash \diamond$ (good environment).	$\Gamma \vdash n : W$ (good message n of type W).	$\Gamma \vdash \kappa : U$ (good context expression κ of privacy type U).
$\Gamma \vdash g :$ gr (good group g).	$\Gamma \vdash P :$ proc$[X, T, U]$ (good process P with mobility type X, exchange type T and privacy type U).	

- $\mathcal{I}$: only ambients of these groups can the context of any ambient of this privacy type be shared with. In other words, only ambients that belong to the groups in $\mathcal{I}$ can *input* (or receive) the context of any ambient of this privacy type. So we use the notation $\mathcal{I}$ for the action *input*.

The intuition behind this privacy type is that an ambient must be able to decide who can read its context, who can share its context with a third-party and who that third-party might be; so as to control both *primary* and *secondary* use of its context information. By convention, we let $*$ denote the set of all groups. So, priv$[\emptyset, \emptyset, \emptyset]$ is the most restricted privacy type (no one can read or share any context information of an ambient of this privacy type) while priv$[*, *, *]$ is the least restricted one (the context of an ambient of this privacy type can be read by everyone and shared with anyone).

Example 1 (Privacy Types). *Let g_1, g_2, and g_3 be three distinct groups. We have the following privacy types:*

1. priv$[\{g_1\}, \emptyset, \emptyset]$*: ambients of group g_1 are allowed to read the context of ambients of this privacy type, but are not allowed to share them with any other ambient.*
2. priv$[\{g_1, g_2\}, \{g_1\}, \{g_3\}]$*: ambients of group g_1 or g_2 are allowed to read the context of ambients of this privacy type, but solely ambients of group g_1 are allowed to share this context with ambients of group g_3. However, ambients of group g_3 are not allowed to share this context further with any other ambient.*
3. priv$[\{g_1\}, \{g_1\}, *]$*: ambients of group g_1 are allowed to read the context of ambients of this privacy type, and to share them with any other ambient. The group g_1 can be thought of as that of a* trusted server *which is allowed to acquire context and share them with the ambients that it trusts, without passing to these ambients the right to share this context further.*

Example 2 (Exchange and mobility types). *In the following types, g is a group:*

- amb$(g)[\curvearrowright,$ shh, priv$[\emptyset, \emptyset, \emptyset]]$*: ambients of this type are of group g, are mobile, quiet (no exchange), and secretive (do not share their context).*
- amb$(g)[\not\curvearrowright, \mathbf{1},$ priv$[*, *, *]]$*: ambients of this type are of group g, immobile, can exchange signals (empty record), and are happy to share their context with anyone.*

For the sake of simplicity, given a privacy type priv$[\mathcal{R}, \mathcal{O}, \mathcal{I}]$, we will refer to the set $\mathcal{R}$ (respectively $\mathcal{O}$, $\mathcal{I}$) as the $\mathcal{R}$-component (respectively $\mathcal{O}$-component, $\mathcal{I}$-component) of that privacy type.

3.2. Typing Rules

We formalise the type system using *typing rules*. The terms that can be typed using these rules are the *well-typed* terms. If n is a name and A a type, we denote by $n : A$ the *assignment* of the type A to the name n; the name n is called the name of that assignment, while A is called the type of that assignment. A *type environment* (aka *type assumption*), denoted by Γ, is a finite set of assignments of types to names, where the names in the assignments are all different.

3.2.1. Type Judgements

A type judgement has the form $\Gamma \vdash G : A$, or $\Gamma \vdash \diamond$, where A is a type and G is a process, a context expression or a name as shown in Table 5. A type judgement $\Gamma \vdash G : A$ asserts that G has the type A under the type assumption Γ; while $\Gamma \vdash \diamond$ means that the type assumption Γ is well-formed (or good) in the sense that the names in assignments are all different. Given a type system, a *valid* type judgement is one that can be proved from the axioms and inference rules of the type system. We now present the axioms and the inference rules of the proposed type system.

Table 6. Context expression typing rules

$$\textbf{(K1)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \texttt{true} : \texttt{priv}[*,*,*]} \quad \textbf{(K2)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \bullet : \texttt{priv}[*,*,*]} \quad \textbf{(K3)}\ \frac{\Gamma \vdash \kappa : U}{\Gamma \vdash \oplus\kappa : U} \quad \textbf{(K4)}\ \frac{\Gamma \vdash \kappa : U}{\Gamma \vdash \neg\kappa : U} \quad \textbf{(K5)}\ \frac{\Gamma \vdash \kappa : U}{\Gamma \vdash \diamond\!\!\!\!\diamond\kappa : U} \quad \textbf{(K6)}\ \frac{\Gamma \vdash \kappa_1 : \texttt{priv}[\mathcal{R}_1, O_1, \mathcal{I}_1] \quad \Gamma \vdash \kappa_2 : \texttt{priv}[\mathcal{R}_2, O_2, \mathcal{I}_2]}{\Gamma \vdash \kappa_1|\kappa_2 : \texttt{priv}[\mathcal{R}_1 \cap \mathcal{R}_2, *, *]}$$

$$\textbf{(K7)}\ \frac{\Gamma \vdash \kappa_1 : \texttt{priv}[\mathcal{R}_1, O_1, \mathcal{I}_1] \quad \Gamma \vdash \kappa_2 : \texttt{priv}[\mathcal{R}_2, O_2, \mathcal{I}_2]}{\Gamma \vdash \kappa_1 \wedge \kappa_2 : \texttt{priv}[\mathcal{R}_1 \cap \mathcal{R}_2, *, *]} \quad \textbf{(K8)}\ \frac{\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]] \quad \Gamma \vdash m : \mu Z.\texttt{amb}(g)[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]]}{\Gamma \vdash n = m : \texttt{priv}[\mathcal{R}, *, *]}$$

$$\textbf{(K9)}\ \frac{\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X, T, \texttt{priv}[\mathcal{R}_1, O_1, \mathcal{I}_1]] \quad \Gamma \vdash \kappa : \texttt{priv}[\mathcal{R}_2, O_2, \mathcal{I}_2]}{\Gamma \vdash n[\kappa] : \texttt{priv}[\mathcal{R}_1 \cap \mathcal{R}_2, *, *]}$$

Table 7. Capability typing rules

$$\textbf{(C1)}\ \frac{\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X', T, U'] \quad \Gamma \vdash P : \texttt{proc}[\curvearrowright, T, U]}{\Gamma \vdash \texttt{in}\ n.P : \texttt{proc}[\curvearrowright, T, U]} \quad \textbf{(C3)}\ \frac{\begin{array}{c}\Gamma, y_1 : W_1, \ldots, y_\ell : W_\ell \vdash P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]] \\ W_{i_j} = \mu Z.\texttt{amb}(g_{i_j})[X_{i_j}, T_{i_j}, \texttt{priv}[\mathcal{R}_{i_j}, O_{i_j}, \mathcal{I}_{i_j}]] \\ 1 \le j \le r \qquad i_j \in \{1, \ldots, \ell\} \qquad j \ne k \Rightarrow i_j \ne i_k \qquad r \le \ell \\ \Gamma \vdash \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \quad T \le \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \quad \mathcal{I}' = (\overset{r}{\underset{j=1}{\Cap}} \mathcal{I}_{i_j}) \cap \mathcal{I}\end{array}}{\Gamma \vdash \theta\ \texttt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}']]}$$

$$\textbf{(C2)}\ \frac{\Gamma \vdash P : \texttt{proc}[\curvearrowright, T, U]}{\Gamma \vdash \texttt{out}.P : \texttt{proc}[\curvearrowright, T, U]} \quad \textbf{(C4)}\ \frac{\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X', T, \texttt{priv}[\mathcal{R}_1, O_1, \mathcal{I}_1]] \qquad \mathcal{R}' = \mathcal{R}_1 \cap \mathcal{R} \quad \Gamma \vdash \theta\ \texttt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]]}{\Gamma \vdash n\,\theta\,\texttt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}', O, \mathcal{I}]]}$$

$$\textbf{(C5)}\ \frac{\begin{array}{c}\Gamma \vdash \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \quad \Gamma \vdash z_1 : W_1 \quad \ldots \quad \Gamma \vdash z_\ell : W_\ell \\ W_{i_j} = \mu Z.\texttt{amb}(g_{i_j})[X_{i_j}, T_{i_j}, \texttt{priv}[\mathcal{R}_{i_j}, O_{i_j}, \mathcal{I}_{i_j}]] \quad O' = (\overset{r}{\underset{j=1}{\Cap}} O_{i_j}) \cap O \\ 1 \le j \le r \quad i_j \in \{1, \ldots, \ell\} \quad j \ne k \Rightarrow i_j \ne i_k \quad r \le \ell \\ \Gamma \vdash P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]] \quad T \le \{l_1 : W_1, \cdots, l_\ell : W_\ell\}\end{array}}{\Gamma \vdash \theta\ \texttt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O', \mathcal{I}]]}$$

$$\textbf{(C6)}\ \frac{\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X', T, \texttt{priv}[\mathcal{R}_1, O_1, \mathcal{I}_1]] \quad \mathcal{R}' = \mathcal{R}_1 \cap \mathcal{R} \quad \Gamma \vdash \theta\ \texttt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}, O, \mathcal{I}]]}{\Gamma \vdash n\,\theta\,\texttt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}', O, \mathcal{I}]]}$$

3.2.2. Context Expression Typing Rules

The typing rules for context expressions are given in Table 6. It follows that the privacy type of the context expression `true` is $\texttt{priv}[*,*,*]$ (the rule K1); rightly so because `true` discloses the name of no ambient. For the same reason, the hole context has the same privacy type (the rule K2). The rules K3, K4 and K5 say that $\neg\kappa$, $\oplus\kappa$ and $\diamond\!\!\!\!\diamond\kappa$ take the type of κ, respectively. The $\mathcal{R}$-component of the type of $\kappa_1|\kappa_2$ and $\kappa_1 \wedge \kappa_2$ is the intersection of the $\mathcal{R}$-components of the types of κ_1 and κ_2 (rules K6 and K7). However, the type of the matching of two ambient names of identical type is a privacy type whose $\mathcal{R}$-component is identical to that of the privacy type of these ambients (the rule K8). If n is an ambient and κ a context expression, then the $\mathcal{R}$-component of the type of $n[\kappa]$ is the intersection of the $\mathcal{R}$-component of the privacy type of n and that of the type of κ (the rule K9).

3.2.3. Capability Typing Rules

The rule C1 in Table 7 says that if n is an ambient of exchange type T, and P a process of mobility type $\curvearrowright$ and exchange type T; then the process '`in` $n.P$' has the mobility type $\curvearrowright$, the exchange T and the same privacy type as P. It is important that the ambient n and the process '`in` $n.P$' have the same exchange type so that an ambient executing that process to move inside n be able to communicate safely with n. Similarly, the rule C2 asserts that any process of the form `out`.P is always of mobility type $\curvearrowright$.

In the rules C3 to C6, θ belongs to the set $\{\uparrow, \downarrow, ::, \epsilon\}$. We also use the notation $\overset{r}{\underset{i=1}{\Cap}} B_i$; where B_i, $1 \le i \le r$ and $0 \le r$ are sets of groups; to denote '*' if $r = 0$ and $\overset{r}{\underset{i=1}{\cap}} B_i$ otherwise. The rule C3 says that the exchange type of a capability `recv` is a record type. If it carries no parameter, then its exchange type is $\mathbf{1}$ (i.e. type of an empty record, $\ell = 0$). If some of those parameters are ambient names, then the $\mathcal{I}$-component of its privacy type must be equal to the intersection of the $\mathcal{I}$-components of the privacy types of these ambients. When the sender of the received messages is specified as in the rule C4, then also the $\mathcal{R}$-component of the privacy type of the `recv` capability must be included in the $\mathcal{R}$-component of the privacy type of that sender. This is because such a capability reads the context (e.g. the location θ) of the sender; hence must be allowed by the sender to do so. The rules C5 and C6 can be explained in a similar way.

Table 8. Process typing rules

$$\textbf{(P1)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : \mathtt{proc}[X,T,\mathtt{priv}[*,*,*]]} \qquad \textbf{(P3)}\ \frac{\Gamma, n : W \vdash P : \mathtt{proc}[X,T,U]}{\Gamma \vdash (\nu\, n : W)P :\ \mathtt{proc}[X,T,U]} \qquad \textbf{(P4)}\ \frac{\Gamma, g : \mathtt{gr} \vdash P : \mathtt{proc}[X,T,U] \qquad g \notin \mathtt{fg}(\mathtt{proc}[X,T,U])}{\Gamma \vdash (\nu\, g : \mathtt{gr})P :\ \mathtt{proc}[X,T,U]}$$

$$\textbf{(P2)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X,T,U]}{\Gamma \vdash\, !P :\ \mathtt{proc}[X,T,U]} \qquad \textbf{(P5)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \quad \Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X,T,U] \qquad g \in \mathcal{R}\cap\mathcal{O}\cap\mathcal{I}}{\Gamma \vdash n[P] :\ \mathtt{proc}[X',T,\mathtt{priv}[*,*,*]]} \qquad \textbf{(P6)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \quad \Gamma \vdash Q : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']]}{\Gamma \vdash P|Q :\ \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}\cap\mathcal{R}',\mathcal{O}\cap\mathcal{O}',\mathcal{I}\cap\mathcal{I}']]}$$

$$\textbf{(P7)}\ \frac{\Gamma \vdash \kappa : \mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1] \quad \Gamma \vdash M.P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}_2,\mathcal{O}_2,\mathcal{I}_2]] \quad \mathcal{R}=\mathcal{R}_1\cap\mathcal{R}_2 \quad \mathcal{O}=\mathcal{O}_1\cap\mathcal{O}_2 \quad \mathcal{I}=\mathcal{I}_1\cap\mathcal{I}_2}{\Gamma \vdash \kappa?M.P :\ \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]} \qquad \textbf{(P8)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \quad \mathcal{R}'\subseteq\mathcal{R} \quad \mathcal{O}'\subseteq\mathcal{O} \quad \mathcal{I}'\subseteq\mathcal{I} \quad T'\leq T}{\Gamma \vdash P : \mathtt{proc}[X,T',\mathtt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']]}$$

3.2.4. Process Typing Rules

The typing rules of processes are depicted in Table 8. The inactivity process $\mathbf{0}$ can take any mobility type and any exchange type; but its privacy type is set to $\mathtt{priv}[*,*,*]$ (the rule P1). Although this privacy type allows the process $\mathbf{0}$ to read the information of any ambient and share it with any other ambient, this will never happen because this process is not able to perform any capabilities. The processes $!P$ and $(\nu\, n : W)P$ take the type of the process P (rules P2 and P3, respectively). Same for the process $(\nu\, g : \mathtt{gr})P$, provided the group g does not occur free in the type of P (the rule P4). The rule P5 is very important and states whether an ambient behaves safely and preserves the privacy of other ambients. So, if n is an ambient of group g and P a well-typed process with the same mobility type and the same exchange type as n such that g belongs to the $\mathcal{R}$-, $\mathcal{O}$- and $\mathcal{I}$-components of the privacy type of P, then $n[P]$ is a well-typed process with the same exchange type as n. If the ambient n's group does not belong to one of these sets, then the process $n[P]$ may violate the privacy of some ambient at run-time. The rules P6 to P8 can be explained in the similar way.

4. Subject Reduction Property

We now have to demonstrate that the proposed type system is consistent with the reduction semantics of CCA (Theorem 4.1). This property is commonly known as the *subject reduction* property of a type system[8]. It states that a well-typed process can only reduce to well-typed processes. A consequence of this property is *type soundness*, asserting that under reduction well-typed processes do not give rise to (i) run-time errors (e.g. exchange of a message of wrong type) –*safety*; (ii) unauthorised disclosure of context information –*privacy*.

Theorem 4.1 (Subject Reduction). *If* $\Gamma \vdash P : A$ *and* $P \rightarrow Q$ *then there exist* $g_1,\ldots,g_\ell$ *such that* $g_1 : \mathtt{gr},\ldots,g_\ell : \mathtt{gr}, \Gamma \vdash Q : A$.

Proof. The proof is done by induction on the derivation of $P \rightarrow Q$. Due to the space limit, we only give the proof for the reduction rule in Eq. 1. Suppose that $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1),\ldots,l_\ell = (y_\ell : W_\ell)\}).P \mid \mathtt{send}(\{l_1 = z_1,\ldots,l_\ell = z_\ell\}).Q : \mathtt{proc}[X,T,U]$. By rule P6, $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1),\ldots,l_\ell = (y_\ell : W_\ell)\}).P : \mathtt{proc}[X,T,U_1]$ and $\Gamma \vdash \mathtt{send}(\{l_1 = z_1,\ldots,l_\ell = z_\ell\}).Q : \mathtt{proc}[X,T,U_2]$ and $U = U_1 \cap U_2$. By rule C3, we must have $T = \{l_1 : W_1,\ldots,l_\ell : W_\ell\}$ and $\Gamma, \{l_1 = (y_1 : W_1),\ldots,l_\ell = (y_\ell : W_\ell)\} \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]]$ and $U_1 = \mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1\cap\mathcal{I}']$, where $\mathcal{I}'$ is the intersection of the $\mathcal{I}$-components of the privacy type of all the y_i that are ambient names; if this intersection is empty then $\mathcal{I}'$ is taken to be '*'. By rule C5, $\Gamma \vdash z_1 : W_1$, ..., $\Gamma \vdash z_\ell : W_\ell$ and $\Gamma \vdash Q : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}_2,\mathcal{O}_2,\mathcal{I}_2]]$ and $U_2 = \mathtt{priv}[\mathcal{R}_2,\mathcal{O}_2\cap\mathcal{O}',\mathcal{I}_2]$, where $\mathcal{O}'$ is the intersection of the $\mathcal{O}$-components of the privacy type of all the z_i that are ambient names; if this intersection is empty then $\mathcal{O}'$ is taken to be '*'. By rule P8 twice, we have $\Gamma \vdash P\{z_1/y_1,\ldots,z_\ell/y_\ell\} : \mathtt{proc}[X,T,U_1]$ and $\Gamma \vdash Q : \mathtt{proc}[X,T,U_2]$. Finally by rule P6, we have $\Gamma \vdash P\{z_1/y_1,\ldots,z_\ell/y_\ell\} \mid Q : \mathtt{proc}[X,T,U]$. ■

5. Related Work

A type system is a powerful technique to check statically that a system cannot violate certain properties at run-time. For instance,[6,8] proposed type systems for ensuring that communications over channels are safe in the pi-calculus and

related languages. Type systems for bigraphs were studied in[9] as an modular approach to devise type systems before transferring them to the family of calculi that the bigraphs model such as the pi-calculus. The mobility types for mobile ambients were introduced in[10]. The concept of group was also used in[11] with a type system to control the mobility of ambients. Unlike these works, we use the concept of group to control the flow of context information among ambients; the privacy type of an ambient specifies the groups of the ambients that are allowed to sense this ambient context information, the groups of the ambients that can output its context information, and the groups of the ambients that can input this context information.

In[12], a type system is proposed for resource access control in a distributed pi-calculus. That type system is based on a notion of location type, which describes the set of resources available to an agent at a location; where a resource is essentially a communication channel. Similar work was done in[13] to control access to tuples spaces in a Linda like language. Likewise,[15] proposed secrecy types to control access to a file system in a variant of the pi-calculus; a secrecy type being assigned a set of the clients that any information of that type can reach. The latter is the closest to our work and their notion of secrecy type can be expressed in our privacy type as $\texttt{priv}[G, G, G]$ where G is the set of the clients that any information of that type can reach. Besides, Privacy-Aware[16] is a sensitive data tracker and eraser that uses type qualifier inference technique in a low-level virtual machine infrastructure to track and eventually erase sensitive data prior to memory deallocation. They use two qualifiers: *sensitive* and *insensitive* that can be thought of as groups assigned to data (i.e. ambient names) in our type system.

6. Conclusion

This paper proposed a novel type system for protecting privacy in ubicomp systems. The type system is based on CCA and comprises mobility types, exchange types and privacy types. The typing rules of the type system were defined; only terms that can be typed using these rules are well-typed. The subject reduction property of the proposed type system was formally established with respect to the reduction semantics of typed CCA. As a consequence, well-typed processes are free from run-time errors caused by an exchange of message of wrong type, and cannot accidentally disclose private information. In future work, we will investigate bisimulation relations for typed CCA to analyse the behaviour of ubicomp systems using a number of case studies.

References

1. Weiser, M.. The Computer for the 21st Century. *Scientific American* 1991;**265**(3):94–104.
2. Schilit, B., Hong, J., Gruteser, M.. Wireless location privacy protection. *Computer* 2003;**36**(12):135–137.
3. Hong, J.I., Ng, J.D., Lederer, S., Landsay, J.A.. Privacy risk models for designing privacy-sensitive ubiquitous computing systems. In: *In Designing Interactive Systems (DIS2004)*. ACM Press; 2004, p. 91–100.
4. Siewe, F., Zedan, H., Cau, A.. The Calculus of Context-aware Ambients. *Journal of Computer and System Sciences* 2011;**77**(4):597–620.
5. Cardelli, L., Gordon, A.D.. Mobile Ambients. *Theoretical Computer Science* 2000;**240**:177–213.
6. Milner, R.. *Communication and Mobile Systems: The π-Calculus*. Cambridge University Press; 1999.
7. Cardelli, L., Ghelli, G., Gordon, A.. Mobility types for mobile ambients. In: *26th International Colloquium on Automata, Languages and Programming (ICALP'99)*; LNCS. Springer; 1999, p. 230–239.
8. Sangiorgi, D., Walker, D.. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press; 2001.
9. Elsborg, E., Hildebrandt, T., Sangiorgi, D.. Type systems for bigraphs. Tech. Rep. TR-2008-110; The IT University of Copenhagen; 2008.
10. Cardelli, L., Ghelli, G., Gordon, A.. Ambient groups and mobility types. In: *IFIP International Conference on Theoretical Computer Science (TCS 2000)*; vol. 1872 of *LNCS*. Springer; 2000, p. 333–347.
11. Coppo, M., Dezani-Ciancaglini, M., Giovannetti, E., Salvo, I.. M^3: Mobility types for mobile processes in mobile ambients. In: *Computing: The Australasian Theory Symposium (CATS'03)*; vol. 78 of *Electronic Notes in Theoretical Computer Science*. 2003, p. 144–177.
12. Hennessy, M., Riely, J.. Resource access control in systems of mobile agents. *Information and Computation* 2002;**173**:82–120.
13. Nicola, R.D., Ferrari, G., Pugliese, R.. Klaim: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering* 1998;**24**(5):315–330.
14. Carriero, N., Gelernter, D., Zuck, L.. Bauhaus linda. In: *Object-Based Models and Languages for Concurrent Systems*; vol. 924 of *LNCS*. Berlin: Springer-Verlag; 1995, p. 66–76.
15. Chaudhuri, A., Abadi, M.. Secrecy by typing and file-access control. In: *19th IEEE Computer Security Foundations Worshop (CSFW'06)*. IEEE Computer Society; 2006, p. 112–123.
16. Ouyang, W.. Privacy-aware: Tracking and protecting sensitive information using automatic type inference. In: *IEEE International Conference on Information Theory and Information Security (ICITIS)*. IEEE Computer Society; 2010, p. 665–668.