Secure Data Flow in a Calculus for Context Awareness

Doina Bucur and Mogens Nielsen

BRICS, Department of Computer Science
University of Aarhus, Denmark
`{doina,mn}@brics.dk`

Abstract. We present a Mobile-Ambients-based process calculus to describe context-aware computing in an infrastructure-based Ubiquitous Computing setting. In our calculus, computing agents can provide and discover contextual information and are owners of security policies. Simple access control to contextual information is not sufficient to insure confidentiality in Global Computing, therefore our security policies regulate agents' rights to the provision and discovery of contextual information over distributed flows of actions. A type system enforcing security policies by a combination of static and dynamic checking of mobile agents is provided, together with its type soundness.

Key words: Ubiquitous Computing, Mobile Ambients, context awareness, security, type system

1 Introduction

The ubiquitous computing systems encourage a constantly changing execution environment for their computing entities. In such settings, context awareness is a computing paradigm in which schemes for context provision and discovery make applications aware of the changes taking place in their computing context, allowing them to gain advantage from context change, instead of employing a middleware layer for hiding the changes from the application.

On the coordinates of the surveys upon context awareness of Chen, Schilit and Abowd [11, 19, 1], *contextual information* or computing context is any piece of information used by an application in order to infer knowledge about other interesting computing entities (objects, persons or places) in its environment. *Primary context* can be divided into (possibly overlapping) categories: resources, such as neighbouring printers or the degree of congestion in the network; user context, meaning an object or person's location or status; physical context, such as temperature or moment in time; history of context, meaning the recording of any primary context over time. More complex conclusions about the environment can be drawn by combining several pieces of primary context into what is denoted by *secondary context*: a source of information is indexed by one type of context, after which the result gets indexed by another.

In order for applications to be able to access contextual information in a dynamic network, *context provision* is the dissemination of contextual information from the host entity to a network neighbourhood, to enlarge its visibility scope; *context discovery* is the locating of the provided context by interested applications. In large networks, the design for context provision and discovery is complicated by the network size: network-wide provision is not feasible, and provision to a locality of players is used instead.

According to Schilit [19], the basic way in which applications make use of the surrounding context is by *contextual information and commands*, i.e. requests for either data or actions, which produce different results depending on the specific context in which they are issued. A context-aware application can also involve *context-triggered actions*, i.e. rules which specify what commands to be automatically executed, given certain properties which the context fulfills. Finally, applications can employ *automatic contextual reconfiguration* to add and remove entire software components in response to certain properties of the context.

Ubiquitous systems are numerous, highly dynamic and their contextual information should be made accessible to a selected subset of the players in the network, features which make such systems difficult to enforce security policies upon. Simple access control upon pieces of contextual information cannot prevent their disclosure by agents collaborating on accessing them; static type checking cannot verify, on its own, that a highly dynamic network respects policies, hence dynamic type checking is called to verify the instances in which agents or code move in the network. Given that such systems are selective, privileges are fine-grained, so that users have different rights upon different pieces of contextual information.

Our calculus models infrastructure-based (as opposed to ad hoc) ubiquitous systems, inherently of hierarchical topology. In such systems, logical partitions are imposed over the network and all communication between mobile entities is mediated by the infrastructure; rooms, floors, buildings, and campuses are such logical cells which act as mediators for context provision, context discovery and general communication for the entities in their scope. Early systems in this category include the Xerox ParcTab [20], Active Badge [21], and GUIDE [12].

Computing entities are modelled by mobile ambients enclosing processes and other ambients, and the topology of the network evolves by the ambients' running of in and out movements. Furthermore, we introduce a modelling of contextual information, distributed through the network over scopes of various sizes and expressed by named macros. The context of an ambient is then the collected contextual information hosted by all ambients enclosing it up until the root ambient, at the ambient's current position. Context provision is performed by an ambient through defining the named macro up in the network to a specified ambient destination to increase the macro's scope. Context discovery is performed by an ambient calling a macro name from a specified ambient and having the call replaced with the macro body, if such a macro exists in the current context. The context changes whenever ambients move and provide or consume contextual information.

The focus of this paper is to study the fulfilling of distributed security policies, in such a setting in which contextual information crosses boundaries in a hierarchical topology. The type of a process is a pair composed by a function recording the effects (provision or discovery) which ambients inner to the process have on other ambients, and by a set of ambient names which are allowed to reside in the process. Any ambient can host security policies in the form of process types, which all processes residing under this ambient should fulfill. This gives, at any given moment in the evolution of the network, for any process sitting in a context composed by a line of ancestor ambients (each with a policy of its own), that the process must adhere to the composition of all ancestor policies. Static type checking verifies that an initial state of the network is well-typed, for dynamic type checking to then verify the incremental movements of definitions and ambients in the network. We show well-typedness to be verified over any sequence of reductions in the system, define errors and show that they cannot appear in a well-typed system.

The rest of this paper is organised as follows: Section 2 presents the syntax and the basic semantics of our calculus, Section 3 the type systems, the notion of well-typedness and the subject reduction theorem, and Section 4 the operational semantics and the notion of type soundness. Finally, Section 5 illustrates a case study modelling a ubiquitous computing infrastructure in a hospital, and Section 6 reviews closely related work and concludes.

2   A Calculus for Context Awareness

In this section we briefly present the syntax and the basic, untyped semantics of the calculus. In the complete syntax from Fig. 1, the nil process 0, parallel composition $P \mid P'$, ambient $a[P]$, name restriction $\nu z\, P$ and movement primitives $in\, a.P$ and $out.P$ are all inherited from and have the same meaning as in the Mobile Ambients calculus [10]; as in the Boxed Ambients calculus [7], there is no *open* capability. From Zimmer's calculus for context awareness [22] we borrow the idea of contextual information as macro definitions residing at ambients. A definition of the basic form $def\, f \triangleright Q\, in\, P$ defines macro f as being the process Q in a floating definition $(f \triangleright Q)$ and continues execution with P, and any call f for this macro would be replaced by its body Q; a simplified semantics for a definition and call are:

$$\text{Def}\quad def\, f \triangleright Q\, in\, P \longrightarrow (f \triangleright Q)\, P \qquad \text{Call}\quad (f \triangleright Q)\, f \longrightarrow Q$$

The decorations τ and G on ambients and floating definitions from Fig. 1 and Table 1 in the following are security and entry policies, respectively, and their syntax and meaning are detailed in Section 3; they are ignored in this section.

Macro definitions come in two flavours: a *one-shot definition* $f \triangleright Q$ is one which is consumed by that macro being called and is suitable for modelling network packets (if one sees data communication as a simple feature of context); a *permanent definition* $!f \triangleright Q$ is one which can be instantiated by any number of macro calls, and in fact behaves like an infinite set of one-shot definitions.

processes	$P ::= 0$	no process
	$\mid \quad P \mid P'$	parallel composition
	$\mid \quad f^a$	macro call, $f \in \mathcal{F}$
	$\mid \quad def^a\, D\, in\, P$	macro definition
	$\mid \quad a_G^\tau\,[P]$	mobile entity, $a \in \mathcal{A}$
	$\mid \quad E\, P$	public definitions
	$\mid \quad \nu z\, P$	name restriction, $z \in \mathcal{F} \cup \mathcal{A}$
	$\mid \quad in\, a.P$	movement in
	$\mid \quad out.P$	movement out
definitions	$E ::= (D)^a$	floating definition, upward
	$\mid \quad (D)^{a,\tau}$	floating definition, downward
	$D ::= F \mid !F$	
	$F ::= f \rhd P$	macro definition

<div align="center">Fig. 1. Syntax</div>

Unlike Zimmer's calculus, definitions and calls of contextual information are not only made by processes to and from their enclosing ambient, but cross multiple ambients' boundaries; hence, definitions and calls are tagged with the identity of the destination and source ambient, respectively: process $def^b f \rhd Q\, in\, P$ publishes macro f at any ambient b in the ancestor ambient line of this process, and process f^b calls macro f from any such ambient. The calls and definitions being tagged with the name of a destination ambient fits the infrastructure setting, in which mobile entities have a degree of knowledge about the identities of servers, gateways and about the protocols in the network, at least such that identities of other points of interest can be provided by calling these.

For this, a process defining macro f to ambient b evolves into a floating definition destined to b:

$$\text{DEF} \quad def^b\, f \rhd Q\, in\, P \longrightarrow (f \rhd Q)^b\, P$$

for the floating definition to travel upwards to destination in fluid movements of the form

$$(f \rhd Q)^b\, P \mid R \equiv (f \rhd Q)^b\, (P \mid R)$$

(included among the structural congruence rules in Table 1) and

$$\text{UP} \quad a\,[(f \rhd Q)^b P] \longrightarrow (f \rhd Q)^b\, a\,[P]$$

(a semantics rule in Section 4). If permanent, a definition at its destination $b\,[(!f \rhd Q)^b P]$ expands single instances upon calls, with $(!F) \equiv (F)(!F)$. When called, a single floating definition moves down from its host ambient to the calling process by the inverse of the upward movements above:

$$\text{DOWN} \quad (f \rhd Q)^b a[P] \longrightarrow a\,[(f \rhd Q)^b P]$$

for the call to be fulfilled at the calling ambient:

$$\text{CALL} \quad (f \triangleright Q)^b \, f^b \longrightarrow Q$$

This scope extension for contextual information follows the idea that context is formed by publishing information from a source to a wider locality of users; it allows for the modelling of context provision and discovery over entire localities of agents, unlike Zimmer's model, which bounds the communication of context within the enclosing ambient of the communicating process. This scheme also effectively models Schilit's contextual information and commands from [19]: a call for a service has a dynamic interpretation varying with context. Furthermore, the intermediate steps a floating definition takes in order to reach a calling ambient gives that either contextual information or its destination can unexpectedly become unreachable with the ambients' changing of location; this fits the profile of highly dynamic networks.

Table 1. Structural congruence is the smallest congruence relation satisfying these rules.

$$
\begin{array}{ll}
P|0 \equiv P & \nu z\,0 \equiv 0 \\
P|Q \equiv Q|P & \nu z\,\nu w\,P \equiv \nu w\,\nu z\,P \\
(P|Q)|R \equiv P|(Q|R) & \nu z\,(P|Q) \equiv P|\nu z\,Q \text{ if } z \notin fn(P) \\
 & \nu z\,(a_G^\tau[P]) \equiv a_G^\tau[\nu z\,P] \text{ if } z \notin fn(a_G^\tau[]) \\
(!F)^{a,\tau} \equiv (F)^{a,\tau}(!F)^{a,\tau} & \nu z\,E\,P \equiv E\,\nu z\,P \text{ if } z \notin fn(E) \\
E_1 E_2 \equiv E_2 E_1 & \\
E\,P|Q \equiv E(P|Q) & \alpha\text{-conversion}
\end{array}
$$

As an example, take a hospital's ubiquitous system supporting collaboration among mobile employees (inspired by [2]); the hospital network infrastructure is ambient hni, and a doctor's personal digital assistant doc currently residing in the operating ward ow updates the network about his location $docloc$ of current value P, so that the tag of any nurse (at any location in the hospital, such as office of) to locate him:

$$hni \left[ow \left[doc \left[def^{hni}!docloc \triangleright P \, in \, 0 \right] \right] \mid of \left[nurse \left[docloc^{hni} \right] \right] \right]$$

The nurse's code cannot execute without the $docloc$ macro available in its context, but after the definition becomes visible the system is:

$$hni \left[(!docloc \triangleright P)^{hni} \left(ow \left[doc \left[\right] \right] \mid of \left[nurse \left[docloc^{hni} \right] \right] \right) \right]$$

and one instance of the definition travels downwards to meet the request:

$$hni \left[(!docloc \triangleright P)^{hni} \left(ow \left[doc \left[\right] \right] \mid of \left[nurse \left[(docloc \triangleright P)^{hni} docloc^{hni} \right] \right] \right) \right]$$

and $(docloc \triangleright P)^{hni} docloc^{hni}$ reduces to P.

We use two sets of names in our syntax: macro names f belong to an infinite set $\mathcal{F}$, and ambient names a belong to an infinite set $\mathcal{A}$. A restriction νz can only

be made upon any name z in $\mathcal{F} \cup \mathcal{A}$, and α-conversion substitutes names from $\mathcal{F} \cup \mathcal{A}$. We adapt the convention: when considering a process, we assume that the bound names of the process are different from its free names, if necessary, after a number of α-conversion steps.

Crucially, the restriction operator ν is the only binder in our syntax; i.e., a macro name in a macro definition is not a binder. Hence, the bound and free names of a process $bn(P)$, $fn(P)$, the substituted process $P\alpha$ and the extrusion of restrictions are defined as is standard.

3   Type Systems and Well-Typedness

We introduce a typing system which specifies the effects which a process in our calculus is allowed to have. A type is two-fold, as depicted in Fig. 2: on one hand, a security policy τ is a function mapping a source ambient name to a destination ambient name and to a set of effects which the source ambient can have upon the destination ambient. An effect is the ability to run either a definition of or a call for a macro name f. On the other hand, the mere presence of an ambient in a process is a securable feature, thus we also enforce entry policies G, as being subsets from the set of ambient names.

Entry policy	$G \subseteq \mathcal{A}$
Effects	$\mathcal{E} = \bigcup_{f \in \mathcal{F}} \{def(f), call(f)\}$
Security policy	$\tau = \mathcal{A} \to (\mathcal{A} \to \mathcal{P}(\mathcal{E}))$

Fig. 2. Entry policies and security policies

As is natural for Mobile-Ambients-based calculi, policies reside at the ambient membrane. We call a *subambient* of a any ambient enclosed by a either directly, or at any inner level. Then, we write $a_G^\tau[P]$ to specify that P should satisfy policies τ and G. If $b \notin G$, then P should not have a subambient b at all. Furthermore, if $\tau(b)(c) \not\ni def(f)$, then P should not have a subambient b directly enclosing a process $def^c f \triangleright Q \, in \, R$; similarly for macro calls.

As an example, the policy τ of the hospital network infrastructure hni_G^τ is such that only employees have access to the patient record of a certain VIP, accessible through the protocol (i.e., macro name) vip directed at hni; thus, for any visitor vis the policy states that $\tau(vis)(hni) \not\ni call(vip)$ and applies throughout the hospital system, without the need of it being multiplied (for example, at the lower, ward level). Also, supposing that one floor $floor3_C^\gamma$ of the hospital is closed to anybody but the hospital's employees, $vis \notin C$, so that no further policies upon vis are needed inside $floor3$.

A fully-permissive entry policy is $\mathcal{A}$, and a fully-permissive security policy is denoted by ω, with

$$\forall b, c \in \mathcal{A} \quad \forall f \in \mathcal{F} \quad \omega(b)(c) \ni def(f) \text{ and } \omega(b)(c) \ni call(f).$$

Two policies can be composed to denote a policy which satisfies both original policies; the composition of entry policies G and H is $G \cap H$, while the composition of security policies τ and σ is the policy $\tau \cap \sigma$, with

$$(\tau \cap \sigma)(b)(c) = \tau(b)(c) \cap \sigma(b)(c).$$

Composition is both commutative and associative, and ω is the identity element, $\omega \cap \tau = \tau$, for all τ. We denote the set of all security policies by $\mathcal{T}$.

We do not detail the matter of defining a particular syntax for policies. We only assume such a syntax to infer a notion of *free names* of policies $^\tau_G$, written $fn\left(^\tau_G\right)$. For all non-free combination of names, the policies are implicitly either fully-permissive or fully-dismissive; then, for the free names, policies specify explicitly either restrictions or allowances, in both cases finitely many.

This notion of free names only needs to satisfy that the set of free ambient names, $fn\left(^\tau_G\right) \cap \mathcal{A}$, and the set of free macro names, $fn\left(^\tau_G\right) \cap \mathcal{F}$, are both finite sets, and that the following conditions are satisfied:

- for entry policies, the set of allowed names of subambients G is either finite or cofinite, relative to the finite set of free ambient names $fn\left(^\tau_G\right) \cap \mathcal{A}$;
- for security policies, any set of allowed definitions and calls $\tau(a)(b)$, $\forall a, b \in \mathcal{A}$ is either finite or cofinite relative to the finite set of free macro names $fn\left(^\tau_G\right) \cap \mathcal{F}$, and this former set is uniformly defined for all non-free ambient names.

Given our hierarchical network topology of mobile agents, each hosting a policy, a process sitting in the scope of a set of ambients should comply with the collected policies of those ambients. The initial state of a system is statically checked for compliance with all the system's policies, and then only individual moves of ambients are checked in the operational semantics described in Section 4.

3.1 A Type System for Active Code

There are two interconnected type systems; the one for *active* processes (i.e. all processes not a macro definition body), with type statements of the form $a^\tau_G \vdash P$ stating that P running at ambient a complies with the type τ, is depicted in Fig. 3. The one for *inactive* processes (i.e. definition bodies, which travel over ambient boundaries before being executed) is discussed subsequently. Finally, Def. 1 at the end of the section defines a process to be well-typed if it is both actively and inactively well-typed.

For the compactness of our typing expressions, we write $?f$ to stand for both permanent $!f$ and one-shot f definitions; similarly, we write $\sigma?$ to stand for both σ and no policy.

$$\text{NULL} \quad a_G^\tau \vdash 0 \qquad \text{PAR} \quad \frac{a_G^\tau \vdash P \quad a_G^\tau \vdash P'}{a_G^\tau \vdash P|P'} \qquad \text{CALL} \quad \frac{\tau(a)(b) \ni call(f)}{a_G^\tau \vdash f^b}$$

$$\text{DEF} \quad \frac{\tau(a)(b) \ni def(f) \quad a_G^\tau \vdash P}{a_G^\tau \vdash def^b\,?f \rhd Q\,in\,P}$$

$$\text{AMB} \quad \frac{b \in G \quad b_{G\cap H}^{\tau\cap\sigma} \vdash P}{a_G^\tau \vdash b_H^\sigma[P]}$$

$$\text{MSG} \quad \frac{a_G^\tau \vdash P}{a_G^\tau \vdash (D)^{b,\sigma?}P} \qquad \text{RES} \quad \frac{a_G^\tau \vdash P}{a_G^\tau \vdash \nu z\,P}\ z \notin fn\left(_G^\tau\right)$$

$$\text{IN} \quad \frac{a_G^\tau \vdash P}{a_G^\tau \vdash in\,b.P} \qquad \text{OUT} \quad \frac{a_G^\tau \vdash P}{a_G^\tau \vdash out.P}$$

Fig. 3. Type system for active code

The interesting rule is AMB: in a top-down checking fashion, for a subambient $b_H^\sigma[P]$ sitting in an ambient a to be actively typed, process P has to be typed with the composed security policy and the intersected entry policy given by the policies of b and a; hence, in any statement of the form $a_G^\tau \vdash P$, P sits in ambient a, τ is the security policy composed of all security policies belonging to ambients ancestor to P, and G is the intersection of their entry policies.

Then, for a process with a call effect f^b sitting in a_G^τ to be actively typed, the condition in the CALL rule is $\tau(a)(b) \ni call(f)$, for the effect to be allowed by the composed policy τ. The same goes for definition effects in the DEF rule, with a further check on the continuation process P following the definition. Entry policies are checked in the AMB rule, in which a subambient b is allowed to run at a_G^τ if $b \in G$.

In the RES rule, $\nu z\,P$ is typed with respect to a_G^τ if z is not one of the free names of policies $_G^\tau$, i.e. those names upon which the policies explicitly specify restrictions or allowances. On the other hand, if z is such a name, z is α-converted to a fresh name. Thus, if e.g. a fresh ambient is created using the restriction operator, at type checking the ambient name is non-free and is checked against implicit policies (e.g. fully-restrictive).

Movement capabilities *in* and *out* are not type checked themselves, but the operational semantics will impose well-typedness conditions for moves, as shown in Section 4. Furthermore, we assume that a system starts in a state without floating definitions, which allows us to only check statically definition bodies only once in their defining process (DEF), and not when floating (MSG).

3.2   A Type System for Inactive Code

For the type checking of definition bodies in Fig. 4, given that definitions travel over ambient boundaries, the check is more complex. Intuitively, a defining process $def^b\,?f \rhd Q\,in\,P$ has as result the definition $(?f \rhd Q)^b$ travelling upwards

to ambient b and then downwards to any calling ambient; in the inactive type system below, it is ensured that when having arrived at b, the body process Q complies with the composed policies of b and its ancestors. Subsequently in Section 4, the remaining down movements will be checked dynamically, to ensure that Q complies with all the policies imposed within b down to the calling ambient.

The *context function* $\mathcal{C}$ records the context, in terms of security and entry policies, with which a definition body should comply when travelling upwards, in order to maintain the well-typedness of the system. The function is totally defined on the ambient names set $\mathcal{A}$ and takes values in pairs of security policies (from the set $\mathcal{T}$) and entry policies (from $\mathcal{P}(\mathcal{A})$). Formally, the type of this function is (written here using a nonstandard notation reflecting our notation for decorating ambients):

$$\mathcal{C} : \mathcal{A} \longrightarrow {}^{\mathcal{T}}_{\mathcal{P}(\mathcal{A})}.$$

Thus, whenever $\mathcal{C}(b) = {}^{\tau}_G$, we have $a\mathcal{C}(b) = a^{\tau}_G$.

The intuition is that, given a definition $def^b f \triangleright Q \, in \, P$, its policy-wise context $\mathcal{C}(b)$ returns the collected policies of ambients above b, so that in order for Q to be run under b, $b\mathcal{C}(b)[Q]$ needs to be well-typed (i.e., both actively and inactively typed against the collected policies $\mathcal{C}(b)$).

As with both security and entry policies, $\mathcal{C}$ has a fully-permissive instantiation Ω with:

$$\Omega(\mathcal{A}) = {}^{\omega}_{\mathcal{A}}.$$

We frequently abuse the notation and write $\mathcal{C}(H) = {}^{\tau}_G$ to state that the value of $\mathcal{C}$ for all elements in set H is ${}^{\tau}_G$. Also, we frequently define an instantiation of such a context function $\mathcal{C}$ by writing updates upon Ω, of the form $\Omega[H \to {}^{\tau}_G]$.

$$\text{Null} \quad \mathcal{C}, a^{\tau}_G \vdasho 0 \qquad \text{Par} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P \quad \mathcal{C}, a^{\tau}_G \vdasho P'}{\mathcal{C}, a^{\tau}_G \vdasho P|P'} \qquad \text{Call} \quad \mathcal{C}, a^{\tau}_G \vdasho f^b$$

$$\text{Def} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P \quad a^{\tau}_G[Q] \text{ well-typed}}{\mathcal{C}, a^{\tau}_G \vdasho def^a \, ?f \triangleright Q \, in \, P} \qquad \frac{\mathcal{C}, a^{\tau}_G \vdasho P \quad b\mathcal{C}(b)[Q] \text{ well-typed}}{\mathcal{C}, a^{\tau}_G \vdasho def^b \, ?f \triangleright Q \, in \, P}$$

$$\text{Amb} \quad \frac{\mathcal{C}[\{a\} \cup G \to {}^{\tau}_G], b^{\tau \cap \sigma}_{G \cap H} \vdasho P}{\mathcal{C}, a^{\tau}_G \vdasho b^{\sigma}_H[P]}$$

$$\text{Msg} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P}{\mathcal{C}, a^{\tau}_G \vdasho (D)^{b,\sigma?}P} \qquad \text{Res} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P}{\mathcal{C}, a^{\tau}_G \vdasho \nu z \, P} \, z \notin fn(\mathcal{C}) \cup fn({}^{\tau}_G)$$

$$\text{In} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P}{\mathcal{C}, a^{\tau}_G \vdasho in \, b.P} \qquad \text{Out} \quad \frac{\mathcal{C}, a^{\tau}_G \vdasho P}{\mathcal{C}, a^{\tau}_G \vdasho out.P}$$

Fig. 4. Type system for inactive code

Inactive type statements have the form $\mathcal{C}, a_G^\tau \vdash\circ P$. This intuitively means that all the definition bodies Q waiting to be activated in processes of the form $def^b f \triangleright Q \, in \, R$ inside P sitting in ambient a will have to be able to run inside the destination ambient: if the destination is the host ambient a itself, then the definition bodies have to comply with a_G^τ, in which, as in the case of the active type system, τ and G are composed by or intersected from all ambients above. On the other hand, for a destination $b \neq a$, the context function $\mathcal{C}(b)$ returns the pair of collected policies from the set of ambients ancestor to b (if b is in P's context) or from the maximal set of ambients which can ever exist above b, were P to move under b.

For consistency, we assume that if one of our systems is not of the form $a_G^\tau[P]$, there exists a unique ambient $world_A^\omega$ with full permissions at the root of the system. The type checking then proceeds top-down, while also collecting contextual policies in a context function $\mathcal{C}$: at top level, there exists no restricting context other than the enclosing root ambient a_G^τ, and a typing statement looks like $\Omega, a_G^\tau \vdash\circ P$; when another ambient is encountered, $\Omega, a_G^\tau \vdash\circ b_H^\sigma[Q]$ if $\Omega\left[\{a\} \cup G \to \frac{\tau}{G}\right], b_{G\cap H}^{\tau\cap\sigma} \vdash\circ Q$ and the context function for ambient b is the updated $\Omega\left[\{a\} \cup G \to \frac{\tau}{G}\right]$, meaning that if a definition in Q is destined to b, it should comply with the composed policies of a and b; if destined to a or any other ambient allowed inside a, it should comply with the policies of a. This last fact is to ensure such a well-typedness, that the dynamic checks at in movements are simple, as will be detailed in Section 4.

We then define well-typedness as being the dual, active and inactive, checking of processes.

Definition 1 (Well-typedness). *A system $a_G^\tau[P]$ is well-typed if $a_G^\tau \vdash P$ and $\Omega, a_G^\tau \vdash\circ P$. A system P without a root ambient is well-typed if $world_A^\omega \vdash P$ and $\Omega, world_A^\omega \vdash\circ P$.*

4   Operational Semantics and Type Soundness

Our operational semantics from Fig. 5 preserves the flavour of reduction rules IN,OUT,STRUCT and CONTEXT from standard Mobile Ambients. Moreover, rules UP and DOWN depict the crossing of ambient borders by definitions floating from their origin to the destination or from the latter to a calling ambient; if these movements are successful, a pair composed by a called definition adjacent to its call reduces to the definition body. In Fig. 5, the *active contexts* **C** are the processes with one hole in active locations (i.e, locations in which a process can suffer a reduction immediately), as in the following:

$$\mathbf{C} ::= [\cdot] \mid \mathbf{C}|P \mid a_G^\tau[\mathbf{C}] \mid E\,\mathbf{C} \mid \nu z\,\mathbf{C}$$

Static checking, discussed in Section 3, verifies that an initial state of the system is well-typed. It includes, in the inactive type checking from Fig. 4, a scheme to verify that even the bodies of those definitions destined to ambients

$$\text{UP} \quad a_G^\tau[(D)^b P] \longrightarrow (D)^b a_G^\tau[P] \qquad a_G^\tau[(D)^a P] \longrightarrow a_G^\tau[(D)^{a,\tau} P]$$

$$\text{DOWN} \quad \frac{a_G^{\sigma \cap \tau}[Q] \text{ well-typed}}{(f \triangleright Q)^{b,\sigma} a_G^\tau[P] \longrightarrow a_G^\tau[(f \triangleright Q)^{b,\sigma \cap \tau} P]}$$

$$\text{DEF} \quad def^a\, D\, in\, P \longrightarrow (D)^a P \qquad \text{CALL} \quad (f \triangleright P)^{a,\tau}\, f^a \longrightarrow P$$

$$\text{IN} \quad \frac{b \in G \quad b_G^\tau \vdash Q \mid R \quad \Omega\left[\{a\} \cup G \to {}_G^\tau\right], b_G^\tau \vdash\!\!\circ\ Q \mid R}{a_G^\tau[P] \mid b_H^\sigma[in\, a.Q \mid R] \longrightarrow a_G^\tau[P \mid b_H^\sigma[Q \mid R]]}$$

$$\text{OUT} \quad a_G^\tau[P \mid b_H^\sigma[out.Q \mid R]] \longrightarrow a_G^\tau[P] \mid b_H^\sigma[Q \mid R]$$

$$\text{STRUCT} \quad \frac{P \equiv P' \quad P \longrightarrow Q \quad Q \equiv Q'}{P' \longrightarrow Q'} \qquad \text{CONTEXT} \quad \frac{P \longrightarrow Q}{\mathbf{C}[P] \longrightarrow \mathbf{C}[Q]}$$

Fig. 5. Operational semantics

which are not currently in the context will safely run at those ambients, whenever they become present. That verifies, for example, that in the setting

$$c_K^\rho\left[a_G^\tau[P] \mid b_H^\sigma\left[in\, a.Q \mid R\right]\right],$$

given that $a \in K$, definitions from $Q \mid R$ destined to a are statically checked against the policies of c_K^ρ. Thus, when b performs the *in* movement, it is sufficient to dynamically check that $Q \mid R$ is well-typed against a's policies, as shown by rule IN.

Furthermore, since static checking insures that the body of a definition is safe to be run in the destination ambient it reached after moving upwards, all that is left for the DOWN movement to dynamically check is the compliance with the collected policies of each ambient down until the calling one. This is ensured by decorating the floating definition $(f \triangleright Q)^a$, when it has reached ambient a_G^τ, with a *signature* equal to τ; with every down movement, the signature is composed with the policy of the newly-crossed ambient, and Q is checked against its signature and the crossed ambient's group policy. This scheme has the flavour of firewall-carrying code.

As an observation, the same static checking scheme, together with the dynamic checking at the DOWN movement, have the side effect that some breakings of policy could be caught by either of these checks (and are caught by the earliest, static checking), if the case is such that the ambient tree in which the definition moves up before reaching the destination also includes the calling ambient. On the other hand, if the upward and downward trees are disjunct, only the dynamic check can verify the definition body.

Furthermore, the reader may have wondered about our less standard approach of modelling the availability of contextual information partially through reduction (rules UP, DOWN in Fig. 5), instead of structural congruence. Our choice is motivated by the fact that we want to model explicitly the operational

behaviour of firewalls in terms of filtering capabilities, in the event of contextual information crossing firewalls during downward moves.

We can now state the subject reduction property, as follows.

Theorem 1 (Subject Reduction). *If P is well-typed and $P \xrightarrow{*} Q$, then Q is well-typed.*

In the behaviour of a system it is considered an error, $P \to err$, if an effect (be it a definition or a call) breaks the security policy of any ambient in its context, or if an ambient's presence breaks the entry policy of any ambient in its context, as in Fig. 6. The superscript τ and the subscript G decorating a context $\mathbf{C}$ are the composition of all security policies above $[\cdot]$, and the intersection of all entry policies above $[\cdot]$, respectively. The composed τ and G for a context $\mathbf{C}$ are defined inductively in Table 2.

$$\text{ERR\_DEF} \quad \frac{(\sigma \cap \tau)(a)(b) \not\supseteq def(f)}{\mathbf{C}_H^\sigma \left[a_G^\tau \left[def^b \, f \rhd Q \, in \, P \mid R \right] \right] \longrightarrow err}$$

$$\text{ERR\_CALL} \quad \frac{(\sigma \cap \tau)(a)(b) \not\supseteq call(f)}{\mathbf{C}_H^\sigma \left[a_G^\tau \left[f^b \mid P \right] \right] \longrightarrow err}$$

$$\text{ERR\_IN} \quad \frac{H \not\supseteq a}{\mathbf{C}_H^\sigma \left[a_G^\tau \left[P \right] \right] \longrightarrow err} \qquad \text{ERR\_STR} \quad \frac{P \equiv P' \quad P' \longrightarrow err}{P \longrightarrow err}$$

Fig. 6. Errors

Table 2. The policies of contexts

$\mathbf{C}_G^\tau$	τ	G
$[\cdot]$	ω	A
$\mathbf{C}_H^\sigma \mid P$	σ	H
$c_K^\rho[\mathbf{C}_H^\sigma]$	$\sigma \cap \rho$	$K \cap H$
$E \, \mathbf{C}_H^\sigma$	σ	H
$(\nu z)\mathbf{C}_H^\sigma$	σ	H

We then state that if a system is well-typed, it can never display an error.

Theorem 2 (Type Soundness). *If P is well-typed then $P \not\xrightarrow{*} err$.*

5   Case Study: Ubiquitous Computing in a Hospital

Consider a ubiquitous computing infrastructure in a hospital, inspired by the AWARE project [2], as in Fig. 7. The hospital network infrastructure *hni* maintains the patient records and keeps track of the location of doctors, nurses,

patient and visitors, all of them carrying PDAs, by having their PDAs announce their presence periodically.

The patient records are read and updated by nurses and doctors using either their tabs or the terminals in the operation ward and patients' ward. Patients and visitors have lower degrees of rights upon accessing–with their tabs or at the terminals–patient records and employee locations.
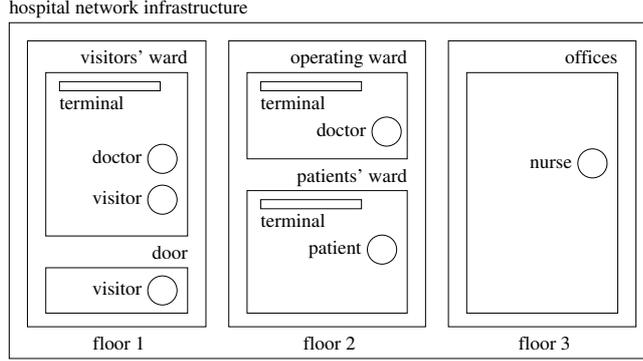

Fig. 7. The hospital system

5.1   The Guessing Visitor

The policy τ of the hospital network infrastructure hni_G^τ is such that only employees have access to the patient record P of a certain VIP, accessible through the protocol (i.e., macro name) vip directed at hni; for any visitor vis the policy states that

$$\tau(vis)(hni) \not\ni call(vip).$$

An overly-curious visitor vis_V^ω who guessed the protocol for accessing the VIP's record would enter through the door $door_\mathcal{A}^\omega$ (i.e., a new ambient vis_V^ω would be run as a subambient of $door$, after being sprouted from a banged definition resident at $door$). The initial, static type checking determines that the system is not well-typed. As an observation, expression $\nu h \left(def^{door} \, !h \triangleright Q \mid h^{door} \, in \, h^{door} \right)$ effectively models $!Q$.

The system, only focused on the door, is formalised as

$$hni_G^\tau [\, (vip \triangleright P)^{hni}$$
$$door_\mathcal{A}^\omega \left[\nu h \left(def^{door} \, !h \triangleright vis_V^\omega \left[vip^{hni} \right] | h^{door} \, in \, h^{door} \right) \right]].$$

We show that its well-typedness depends on at least the policy condition which is not met, $\tau(vis)(hni) \not\ni call(vip)$; the other conditions for its well-typedness, possibly satisfied, are not depicted. The system is well-typed if it is also inactively well-typed, as in the derivation tree in Fig. 8. Given that the well-typedness of the system depends on a condition which is false, the error is raised at this stage.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\tau(vis)(hni) \ni call(vip)}{vis^\tau_{G \cap V} \vdash vip^{hni}}}{door^\tau_G \vdash vis^\omega_V \left[vip^{hni}\right]}}{door^\tau_G \left[vis^\omega_V \left[vip^{hni}\right]\right] \ \text{well-typed}}}{\Omega\left[\{hni\} \cup G \to \frac{\tau}{G}\right], door^\tau_G \vdash\circ def^{door} \, !h \triangleright vis^\omega_V \left[vip^{hni}\right] \mid h^{door} \, in \, h^{door}}}{\Omega\left[\{hni\} \cup G \to \frac{\tau}{G}\right], door^\tau_G \vdash\circ \nu h \, \left(def^{door} \, !h \triangleright vis^\omega_V \left[vip^{hni}\right] \mid h^{door} \, in \, h^{door}\right)}}{\Omega, hni^\tau_G \vdash\circ door^\omega_{\mathcal{A}} \left[\nu h \, \left(def^{door} \, !h \triangleright vis^\omega_V \left[vip^{hni}\right] \mid h^{door} \, in \, h^{door}\right)\right]}}{\Omega, hni^\tau_G \vdash\circ (vip \triangleright P)^{hni} \, door^\omega_{\mathcal{A}} \left[\nu h \, \left(def^{door} \, !h \triangleright vis^\omega_V \left[vip^{hni}\right] \mid h^{door} \, in \, h^{door}\right)\right]}$$

with side condition $h \notin fn\left(\frac{\tau}{G}\right)$

Fig. 8. The derivation tree for the guessing visitor scenario

5.2   The Conspiring Nurse

Have a nurse $nurse^\omega_N$ who agreed to conspire with the overly-curious visitor vis^ω_V. They plan on an indirect access scheme to the record vip^{hni}, for the visitor. The visitor's actions will be inconspicuous: the visitor residing in the visitors' ward vw and the nurse in her office of agree upon a new, private service (i.e., macro name) key, assuming that the nurse is allowed to define key, $key \notin fn(\tau)$. The nurse makes key give access to the VIP record's service name: $def^{hni} \, key \triangleright vip^{hni} \, in \, 0$, for then the visitor to call key^{hni}.

The hospital system is now:

$$hni^\tau_H \big[\, (!vip \triangleright P)^{hni} \, \nu \, key \, ($$
$$floor1^\alpha_{\mathcal{A}} \left[vw^\sigma_{\mathcal{A}} \left[vis^\omega_V \left[key^{hni}\right]\right]\right] \mid$$
$$floor3^\gamma_C \left[of^\pi_O \left[nurse^\omega_N \left[def^{hni} \, key \triangleright vip^{hni} \, in \, 0\right]\right]\right]\big)\big)$$

Static checking on this state of the system passes without errors. The dynamic checking at runtime raises the error the moment in which the nurse's definition is about to enter the visitor's ambient. The reduction steps, up until the raising of the error, are:

$$\xrightarrow{*} \nu \, key \, hni^\tau_H \big[\, \left(key \triangleright vip^{hni}\right)^{hni} (!vip \triangleright P)^{hni}$$
$$floor1^\alpha_{\mathcal{A}} \left[vw^\sigma_{\mathcal{A}} \left[vis^\omega_V \left[key^{hni}\right]\right]\right] \mid$$
$$floor3^\gamma_C \left[of^\pi_O \left[nurse^\omega_N \left[\,\right]\right]\right]\big]$$

$$\xrightarrow{*} hni^\tau_H \big[\, (!vip \triangleright P)^{hni}$$
$$floor1^\alpha_{\mathcal{A}} \left[vw^\sigma_{\mathcal{A}} \left[\nu \, key \, \left(\left(key \triangleright vip^{hni}\right)^{hni, \tau \cap \alpha \cap \sigma} vis^\omega_V \left[key^{hni}\right]\right)\right]\right] \mid$$
$$floor3^\gamma_C \left[of^\pi_O \left[nurse^\omega_N \left[\,\right]\right]\right]\big]$$

in which the down movement

$$\left(key \triangleright vip^{hni}\right)^{hni, \tau \cap \alpha \cap \sigma} vis^\omega_V \left[key^{hni}\right] \longrightarrow vis^\omega_V \left[\left(key \triangleright vip^{hni}\right)^{hni, \tau \cap \alpha \cap \sigma} key^{hni}\right]$$

is allowed only if $vis_V^{\tau \cap \alpha \cap \sigma} \left[vip^{hni} \right]$ is well-typed, a condition which depends on $\tau(vis)(hni) \ni call(vip)$:

$$\frac{\dfrac{vis_V^{\tau \cap \alpha \cap \sigma} \left[vip^{hni} \right] \text{ well-typed}}{vis_V^{\alpha \cap \tau} \vdash vip^{hni}}}{\tau(vis)(hni) \ni call(vip)}$$

5.3   The Wandering Visitor

Have the hospital policies devise a scheme to limit visitors from loading their own services (say, named key) throughout the hospital. For this, the first floor $floor1_{\mathcal{A}}^{\alpha}$, enclosing the door and the visitors' ward, has $vis \in A$ and α allowing visitors to publish services, but only up to the floor's level: $\alpha(vis)(floor1) \ni def(key)$, but $\alpha(vis)(hni) \not\ni def(key)$. The second floor still allows patients to be present during visiting hours, but imposes that they shouldn't publish anything while there, to any destination: both $\beta(vis)(floor2) \not\ni def(key)$ and $\beta(vis)(hni) \not\ni def(key)$. There should never be a visitor on the third floor, $vis \notin C$, hence γ poses no further restrictions upon the visitor's actions.

Have the visitor vis_V^{ω} planning to tour the hospital's three floors in search of spots to load the system with his key service. A try at publishing key at hni from the visitor's ward:

$$hni_H^{\tau}[floor1_{\mathcal{A}}^{\alpha} \left[vw_{\mathcal{A}}^{\sigma} \left[vis_V^{\omega} \left[def^{hni} key \triangleright P \, in \, 0 \right] \right] \right]]$$

is signalled at static checking, since the active well-typedness of vis_V^{ω} in this context of security policies depends on a security condition which doesn't hold:

$$\frac{vis_{H \cap V}^{\tau \cap \alpha \cap \sigma} \vdash def^{hni} key \triangleright P \, in \, 0}{\alpha(vis)(hni) \ni def(key)}$$

The visitor would, however, be able to load the first floor (which acts like a sandbox for his definitions) with his service, if performing $def^{floor1} key \triangleright P \, in \, 0$.

If still undeterred in his trials, he could walk to the second floor having in mind to try defining $def^{floor2} key \triangleright P \, in \, 0$:

$$hni_H^{\tau}[floor1_{\mathcal{A}}^{\alpha} \left[vw_{\mathcal{A}}^{\sigma} \left[\right] \right] \, | $$
$$vis_V^{\omega} \left[in \, floor2.def^{floor2} key \triangleright P \, in \, 0 \right] \, | $$
$$floor2_B^{\beta} \left[\right]]$$

In this case, the operational semantics for the in movement raises the error when dynamically checking that:

$$\frac{vis_B^{\beta} \vdash def^{floor2} key \triangleright P \, in \, 0}{\beta(vis)(floor2) \ni def(key)}$$

and the same happens if moving to the third floor, upon the condition $vis \notin C$ at the dynamic checking for in.

6   Related Work, Conclusions and Future Work

There are few direct formal models of context awareness in the presence of mobility. Among these, Birkedal et al. [3] propose a complex model of context awareness able to model both the usual reconfigurations of the context, and queries upon context; noting that context queries cannot be naturally modelled with one bigraphical reactive system (BRS), it proposes a solution (called a Platographical model), such that its expressivity suits well sophisticated real-world context-aware systems. Braione [5] builds contextual reactive systems (CRS) upon reactive systems, RS. The difference between a CRS and a RS is the presence of a function which captures an association between elementary rules and their allowed reaction contexts, so that a CRS can express inhibitor and enabler factors for interaction. Both models are yet to achieve full results in studying behavioural equivalences and proving program properties.

Roman, Julien and Payton [16, 18] build a language-based model of context-aware systems under mobility, an interesting feature of which is the fact that agents have as context the exposed (not private) variables of other agents. The work also has a limited associated proof logic, with program properties being expressed as predicate relations whose validity can be derived. Kjærgaard and Bunde-Pedersen's Conawa calculus [17] models context using several context trees, one for each category of context information (e.g., one for location information, one for activities and one for printers). An ambient entity will have a pointer-like presence in one or more trees, with the usual in/out ambient capabilities extended for mobility in multiple contexts.

Furthermore, we found inspiration in work on securing information flow in programming languages, such as Boudol's typing of information flow [4] in a multi-threaded ML-like language, when declassifying information for legal users (a stronger type system with flow policies, also guarding against termination leaks).

A number of type systems were introduced for Mobile Ambients: Cardelli, Ghelli and Gordon [9], Coppo et al. [13], Gorla, Hennessy and Sassone [14], Bugliesi, Castagna and Crafa [8], among others, introduce message exchange types, the typing of capabilities and actions, type-level groups of ambient names (effectively giving policies for crossing, opening, exchanging messages), and various security policies as membranes at ambient boundaries. Of particular interest is Gorla and Pugliese's enforcing of security policies via types in μKLAIM [15] for its fine-grained security features, assigning different privileges to users over different resources in a flat topology of networks.

Our calculus aims at capturing a notion of context awareness in infrastructure-based ubiquitous systems over a well-understood and applicable formalisation such as Mobile Ambients; we also put to use our background in designing protocols for ubiquitous systems (Bucur and Bardram, [6]) to ground this work in practice. Unlike the standard ambient communication scheme, we model context and its communication by exposing and calling named macros across ambient boundaries and policies, extending Zimmer's [22] flat context communication model. We design for a model of dynamic context with contextual information

being macros distributed over varying network scopes; we follow the previous work in the field of designing for security with Mobile Ambients and apply security policies at ambients' membranes, limiting the capabilities enclosed processes can exhibit, to then type processes in regard to the hierarchy of policies enclosing them. Also, we keep the fine-grained policies on the lines of μKLAIM, only applied over our hierarchical topology of locations.

Our calculus is applicable for modelling and reasoning upon a fraction of the aspects of context-aware computing, in systems deployed over cell-based, hierarchical topologies. The model can aid the understanding of the workings of context in such mobile systems, and guide the implementation of an added software layer for security. Moreover, we feel that the basic ideas of representing contextual information, its communication and its use are applicable to a greater extent; thus, as part of the future work we intend to focus on modelling context awareness in ad hoc ubiquitous systems.

Acknowledgements The authors wish to thank the anonymous reviewers for their comments on improving this paper.

References

1. Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.
2. Jakob E. Bardram and Thomas R. Hansen. The AWARE architecture: supporting context-mediated social awareness in mobile cooperation. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 192–201. ACM Press, 2004.
3. L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems. In *Foundations of Software Science and Computation Structures (FOSSACS) 2006*, number 3921 in Lecture Notes in Computer Science, March 2006.
4. Gerard Boudol. On typing information flow. *Lecture Notes in Computer Science (LNCS)*, 3722:366–380, 2005.
5. Pietro Braione. Operational congruences for contextual reactive systems. Technical Report Technical report 2004.33, DEI, Politecnico di Milano.
6. Doina Bucur and Jakob E. Bardram. Resource discovery in activity-based sensor networks. *Mobile Networks and Applications (MONET)*, 12(2-3):129–142, 2007.
7. Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *TACS '01: Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software*, pages 38–63. Springer-Verlag, 2001.
8. Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Reasoning about security in mobile ambients. *Lecture Notes in Computer Science*, 2154:102–120, 2001.
9. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the ambient calculus. *Inf. Comput.*, 177(2):160–194, 2002.
10. Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.

11. Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Dartmouth Computer Science Technical Report TR2000-381, 2000.
12. Keith Cheverst, Nigel Davies, Keith Mitchell, and Adrian Friday. The role of connectivity in supporting context-sensitive applications. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 193–207. Springer-Verlag, 1999.
13. Mario Coppo, Mariangiola Dezani-Ciancaglini, Elio Giovannetti, and Ivano Salvo. M: Mobility types for mobile processes in mobile ambients. *Electronic Notes in Theoretical Computer Science*, 78, 2002.
14. D. Gorla, M. Hennessy, and V. Sassone. Security policies as membranes in systems for global computing. *Logical Methods in Computer Science*, 1 (1:3):1–23, 2005.
15. D. Gorla and R. Pugliese. Enforcing security policies via types. *LNCS Security in Pervasive Computing*, 2802:86–100, 2004.
16. Christine Julien, Jamie Payton, and Gruia-Catalin Roman. Reasoning about context-awareness in the presence of mobility. *Electr. Notes Theor. Comput. Sci.*, 97:259–276, 2004.
17. Mikkel B. Kjærgaard and Jonathan Bunde-Pedersen. Towards a formal model of context awareness. In *First International Workshop on Combining Theory and Systems Building in Pervasive Computing 2006 (CTSB 2006)*, 2006.
18. Gruia-Catalin Roman, Christine Julien, and Jamie Payton. A formal treatment of context-awareness. In *Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004)*, pages 12–36. Lecture Notes in Computer Science 2984, Springer, 2004.
19. Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, 1994.
20. Bill N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, 1995.
21. Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. Technical Report 92.1, Olivetti Research Ltd. (ORL), 1992.
22. Pascal Zimmer. A calculus for context-awareness. Technical Report BRICS Report Series RS-05-27.