Electronic Notes in Theoretical Computer Science 66 No. 3 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume66.html 22 pages

Information Flow Security in Boxed Ambients

Silvia Crafa[1] Michele Bugliesi [1]

Dipartimento di Informatica
Università Ca' Foscari
Venezia, Italy

Giuseppe Castagna [2]

Département d'Informatique
École Normale Supérieure
Paris, France

Abstract

We study the problem of secure information flow for Boxed Ambients in terms of non-interference. We develop a sound type system that provides static guarantees of absence of unwanted flow of information for well typed processes. Non-interference is stated, and proved, in terms of a typed notion of contextual equivalence for Boxed Ambients akin to the corresponding equivalence defined for Mobile Ambients.

1 Introduction

The calculus of Boxed Ambients [4] is a novel process calculus derived from Mobile Ambients [8] to provide finer grained abstractions for resource protection and access control in systems of distributed and mobile agents.

In Mobile Ambients, abbreviated MA, agents are processes of the form $n[P]$, representing the ambient, named n, executing the process P. Processes can be composed in parallel, as in $P \mid Q$, exercise a *capability*, as in $M.P$, declare local names as in $(\boldsymbol{\nu} n)P$, or simply do nothing as in $\mathbf{0}$. Ambients may be nested to form a tree structure that can be dynamically reconfigured by exercising the capabilities in, out and open. As an example, the system

$$k[\mathsf{in}\ n.P_1 \mid m[\mathsf{out}\ n.P_2]] \mid n[\mathsf{open}\ k.Q]$$

contains two ambients, k and n, running in parallel. The system may evolve as follows. First, ambient k may migrate to n by exercising the capability "$\mathsf{in}\ n$":

[1] Email: {silvia,michele}@dsi.unive.it
[2] Email: Giuseppe.Castagna@ens.fr

$n[\mathsf{open}\ k.Q \mid k[P_1 \mid m[\mathsf{out}\ n.P_2]]]$. Now n may "dissolve" k unleashing its contents: $n[Q \mid P_1 \mid m[\mathsf{out}\ n.P_2]]$. Finally, m may exit n: $n[Q \mid P_1] \mid m[P_2]$. In addition, ambients and processes may communicate. In MA, communication is anonymous, and happens inside ambients. The system $(x)P \mid \langle M \rangle$ represents the parallel composition of two processes, the output process $\langle M \rangle$ "dropping" the message M, and the input process $(x)P$ reading the message M and continuing as $P\{x := M\}$.

The calculus of Boxed Ambients, henceforth BA, is a variant of MA from which it inherits the primitives in and out (but not open) for mobility with exactly the same semantics. As for communication, besides local exchanges, BA relies on an additional set of primitives that provide for the exchange of values across ambient boundaries, between parent and child. Syntactically, this is achieved by means of tags that specify the *location* with which the exchange should take place: as an example, $(x)^n P$ indicates an input from a child ambient name n, while $\langle M \rangle^\uparrow$ is an output to the parent ambient (in the latter case we speak of an *upward* exchange). The semantics of parent-child exchanges is defined by the following reductions [3] :

$$(x)^n P \mid n[\langle M \rangle^\uparrow Q] \rightarrow P\{x := M\} \mid n[Q]$$

$$\langle M \rangle^n P \mid n[(x)^\uparrow Q] \rightarrow P \mid n[Q\{x := M\}]$$

This semantics of communication yields, as a byproduct, a direct interpretation of the local and upward anonymous channels of an ambient as that ambient's "resource space": the local channel is private to the ambient, whereas the upward channel is available for access by clients. By relying on this interpretation, one can formalize a precise notion of resource access, namely: $(x)^n P$ is a read access to n, whereas $\langle M \rangle^n P$ is a write access.

In [5] we showed that BA provides an effective framework for resource access security: specifically, we used a typed version of BA to model multilevel Mandatory Access Control (MAC) policies, including both military (no read-up, no write-down) and commercial (no read-up, no write-up) security [4] .

Boxed Ambients and Information Flow Security

The type system we defined in [5] was targeted at resource access control, and specifically designed to protect resources, viz. channels, from undesired uses by unauthorized clients. Here, we change perspective, and focus on a different analysis that targets information flow. To motivate the change in perspective, consider the following example, where ℓ is a "low-level" ambient and h a "high-level" one:

$$\ell[(x)^h P \mid h[\langle M \rangle^\uparrow Q \mid R]] \qquad (1)$$

[3] These are *not* the reductions we introduced in [5,4]. The choice of the present new semantics is motivated in Section 2.

[4] Although the resource access control policies we study in [5] are based on a different reduction semantics, those techniques adapt smoothly to the new semantics (cf. Section 2 for a discussion).

In the system of [5] the attempt by the process enclosed in the low-level ambient ℓ to read from h is classified as a *read-up*, and therefore rejected as "insecure" by both military and commercial security, regardless of the information content of M. On the other hand, if the expression M is public, i.e., low-level, there is no reason for disallowing the read access: a piece of data is flowing from high to low, but the flow is "secure" as the piece of data carries a "low" information content.

The goal of the present paper is to provide static safeguards against "unsafe" flow of information. A first, rather intuitive, notion of information flow may directly be related to the flow of data: a system is "secure" if no high-level data flows from high-level to low-level principals. This form of secure information flow is easily accounted for, and enforced, by a static control over the transport layer used for data communication, viz channels. If we classify data and channels according to their security levels, absence of this form of "explicit" flow of information can be guaranteed by requiring that:

($\star$) High-level data be only communicated along high-level channels, and high-level channels be only located within high-level subjects.

A subtler, and more interesting, notion of information flow security is related to the presence of implicit flow of information, resulting from indirect ways of transmitting information (namely, covert channels) via system-wide side effects. To illustrate, consider the following specialization of the system (1) above, where P is "low level":

$$\ell[(x)^h\langle N\rangle^\uparrow \mid h[\langle M\rangle^\uparrow]] \;\mid\; (x)^\ell P \tag{2}$$

Assuming that M and N are low-level values, there is no direct flow in this system. However, a covert channel is established between P and the ambient h, as P is unleashed by an exchange that depends on the presence of the high ambient h, and the very presence (or absence) of a high-level ambient can be assimilated to a bit of high-level information that, in the system in question, flows downwards.

Information Flow Security and Non-interference

Defining what is exactly meant by (implicit) information flow can be hard (perhaps impossible), and various authors have instead relied on *non-interference*, a concept of easier formalization which implies absence of flow.

The notion of non-interference was first proposed by Goguen and Meseguer [14] for deterministic state machines. The idea is to determine whether in a given system the "inputs" of high level subjects (or "users") may influence, i.e. interfere with, the "outputs" of low level subjects. If the latter are invariant on the former, then the system is decreed interference free.

Non-interference was later [13] reformulated in a CCS-like process calculus as the so-called *Non Deducibility on Composition* (NDC) property, which implies that low-level observers are insensitive to the presence of high-level components (sources) in the system. Here we take the same approach, and rephrase the NDC property to capture ambient-based specific aspects of computation, namely, locality and mobility.

Overview

Our technique for providing guarantees of non-interference in BA is based on static typing. We rely on types both to formalize the notions of high and low data and processes, and to define the relation of process equivalence underlying the definition of non-interference.

The type system is based on ambient and capability types akin to those employed in companion type systems for MA. In addition, the types of our type system carry security annotations that define the security clearance of values and processes. Processes have the clearance of their enclosing ambient, while values (i.e. capabilities and names) are assigned security levels as follows: names have the security level associated to their type, while capabilities are decreed "low-level" data, based on the observation that capabilities do not disclose their target ambient names, and hence provide rather limited control over such names.

Having partitioned data and processes into "high" and "low", we then single out the set $\mathcal{H}$ of *high level sources* as the set of all processes that can only produce high-level "inputs", where an "input" corresponds to the presence, at top level, either of a communication, or an ambient, or a mobility action. Again, this notion is formalized with the help of the type system (see Section 3).

As the next step of our formalization, we introduce a relation of behavioral equivalence to compare processes. This relation is a typed version of the equivalence relation introduced in [9] for MA: a *contextual equivalence* that equates two processes if and only if they admit the same elementary observations whenever they are inserted inside any arbitrary, but well-typed, enclosing context. Our *observability predicate* is akin to the one studied in [9], but refined to capture the core form of interaction between Boxed Ambients, namely, the ability for an ambient to exchange values along its upward channel. We thus say that a process P exhibits a name n if P (reduces, in any number of steps, to a process that) contains an ambient n that may accept interactions with the external environment, that is if P (or any of the processes it reduces to) is structurally equivalent to $(\boldsymbol{\nu} m_1)\dots(\boldsymbol{\nu} m_k)(n[\langle M\rangle^{\uparrow}P' \mid Q'] \mid Q'')$ where $n \notin \{m_1,\dots,m_k\}$. Even though this notion of observation is specifically focused on communication, ambient mobility is still observed, indirectly, via its consequences on upward communications, as the following example illustrates (the presence of a high level ambient h triggers the upward communication of the low-level ambient ℓ):

$$(x)^{\ell}P \mid \ell[\mathsf{in}\ h.\mathsf{out}\ h.\langle M\rangle^{\uparrow}] \mid h[\,]$$

Finally, we introduce the notion of contextual equivalence induced by a *low level observation*: two (well-typed) processes P and Q are equivalent, $P \cong_{\mathsf{L}} Q$, if whenever they are inserted inside an arbitrary (well-typed) context, they exhibit the same *low level* names. Based on that, we can phrase the NDC property of [13] for BA as:

($\star\star$) A process P is *interference free* if and only if $\forall H \in \mathcal{H} \quad P \mid H \cong_{\mathsf{L}} P$.

As in [13] non interference of P is checked only against high-level sources that appear in parallel with P. This is rather natural in that context, since the topology of CCS processes is completely flat. On the contrary, in BA ambients may be nested arbitrarily and, consequently, a high-level process interacting with P may also (i) enclose P or (ii) be enclosed within P. It would then appear that our definition of non-interference should be generalized to capture these additional cases. Indeed, however, the definition, as given, does address these cases as ambients running in parallel may nest arbitrarily as a result of mobility.

Main results and paper plan

The main contributions of the paper are (i) the definition of a sound type system for the new version of BA and, more importantly, (ii) a proof that well-typed processes are indeed interference free, in the sense we just outlined. The non-interference proof builds on the technical tools developed in [9] by Cardelli and Gordon for MA, adapting them to BA, and relies critically on the choice of contextual equivalence as the underlying equivalence relation. In fact, as we discuss in Section 6, our present results do not extend to finer equivalence relationships, such as barbed congruence [19].

The paper continues as follows. In Section 2 we present a new version of Boxed AmbientsCalculus, that differs from [4] in the semantics of its communication model. The resulting calculus has a simpler presentation and its finer-grained control over ambient interactions more naturally enables the development of an algebraic theory and a security assessment. In Section 3 we describe a sound type system for BA, whose well typed processes are proved in Section 4 to be interference free. Sections 5 and 6 are dedicated to related work and conclusions.

2 Boxed Ambients

In this section we review the syntax of Boxed Ambients from [4], and we present a new reduction semantics, borrowed from [11] (where it was first introduced for the Seal Calculus), and defined in terms of new rules for communication across ambient boundaries. The new calculus still adheres the principle of resource locality distinctive of the original calculus, while at the same time providing ambients with full control of exchanges they may have with their children.

2.1 Syntax

The syntax of the typed calculus is defined by the following productions:

Expressions

M	$::=$	$m - q$	names
	$\mid$	$x - z$	variables
	$\mid$	$\mathsf{in}\ M$	enter M
	$\mid$	$\mathsf{out}\ M$	exit M
	$\mid$	$M.M$	path

Locations

η	$::=$	M	names, variables
	$\mid$	$\uparrow$	parent ambient
	$\mid$	$\star$	local

Processes

$P ::=$	$\mathbf{0}$	stop
$\mid$	$M.P$	action
$\mid$	$(\boldsymbol{\nu} n{:}W)P$	restriction
$\mid$	$P \mid P$	composition
$\mid$	$M[P]$	ambient
$\mid$	$!P$	replication
$\mid$	$(x_1{:}W_1, .., x_k{:}W_k)^\eta P$	input
$\mid$	$\langle M_1, \dots, M_k\rangle^\eta P$	output

Expression Types

$W ::=$	$\mathsf{amb}[E]$	ambient
$\mid$	cap	capability

Exchanges

$E, F ::=$	shh	no exchange
$\mid$	$W_1 \times \dots \times W_n$	exchange

Process Types

$T ::= [E, F]$ composite exchange

As in MA, processes can be named, as in $n[P]$, be composed in parallel and replicated, exercise a capability in or out, declare local names, do nothing or exchange values. Input processes may read a value locally, as in $(x{:}W)^\star P$, from a subambient named n, as in $(x{:}W)^n P$, or from the enclosing context: $(x{:}W)^\uparrow P$. Corresponding primitives are provided for output. As usual, the syntax allows the formation of meaningless process forms such as $\mathsf{in}\ (\mathsf{out}\ m)$ or $(\mathsf{out}\ n)[P]$: these terms may arise as a result of reduction, but only for ill-typed terms. We use a number of notation conventions. We use $m, n, \dots q$ to range over *names*, x, y, z over variables, and a, b, c over both. We write $(\tilde{x}{:}\tilde{W})P$ for $(x_1{:}W_1, \dots, x_k{:}W_k)P$, $\langle\tilde{M}\rangle$ for $\langle M_1, \dots, M_k\rangle$, and $(\boldsymbol{\nu}\tilde{p})P$ for $(\boldsymbol{\nu}p_1)\dots(\boldsymbol{\nu}p_k)P$. As usual we omit trailing dead processes, writing M for $M.\mathbf{0}$, $\langle\tilde{M}\rangle$ for $\langle\tilde{M}\rangle\mathbf{0}$, and $a[\,]$ for $a[\mathbf{0}]$. We also omit type annotations in restrictions and input prefixes when they are not important. Finally, the superscript $\star$ denoting local communication, is omitted.

2.2 Dynamic Semantics

The definition of the sets of free names *fn*(P) and free variables *fv*(P) of a process P is straightforward, once we know that the former are bound by restrictions and the latter by input prefixes. We identify processes up to α-renaming of bound names and variables. Furthermore, assuming that $\tilde{x}$ and $\tilde{M}$ stand for $x_1, \dots, x_k$ and $M_1, \dots, M_k$, we write $P\{\tilde{x} := \tilde{M}\}$ to indicate the capture-avoiding, simultaneous, substitution of M_i for x_i within P.

Structural congruence is defined as the least congruence relation that is a commutative monoid for $\mathbf{0}$ and $\mid$ and closed under the following rules

(Res Dead)	$(\boldsymbol{\nu} m)\mathbf{0} \equiv \mathbf{0}$	
(Path Assoc)	$(M.M').P \equiv M.(M'.P)$	
(Repl)	$!P \equiv !P \mid P$	
(Res Res)	$(\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P \equiv (\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P$	$n \neq m$
(Res Par)	$(\boldsymbol{\nu} m)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} m)Q$	$m \notin fn(P)$
(Res Amb)	$(\boldsymbol{\nu} m)a[P] \equiv a[(\boldsymbol{\nu} m)P]$	$m \neq a$

Structural congruence is functional to the definition of the reduction relation of Figure 1.

Evaluation Contexts $\quad E ::= - \mid (\boldsymbol{\nu} n{:}W)E \mid P \mid E \mid E \mid P \mid n[E]$

(ENTER) $\quad a[\mathsf{in}\ b.P \mid Q] \mid b[R] \rightarrow b[a[P \mid Q] \mid R]$

(EXIT) $\quad a[b[\mathsf{out}\ a.P \mid Q] \mid R] \rightarrow b[P \mid Q] \mid a[R]$

(LOCAL) $\quad (\tilde{x})P \mid \langle \tilde{M} \rangle Q \rightarrow P\{\tilde{x} := \tilde{M}\} \mid Q$

(INPUT a) $\quad (\tilde{x})^a P \mid a[\langle \tilde{M} \rangle^{\uparrow} Q \mid R] \rightarrow P\{\tilde{x} := \tilde{M}\} \mid a[Q \mid R]$

(OUTPUT a) $\quad \langle \tilde{M} \rangle^a P \mid a[(\tilde{x})^{\uparrow} Q \mid R] \rightarrow P \mid a[Q\{\tilde{x} := \tilde{M}\} \mid R]$

(STRUCT) $\dfrac{P \equiv P' \quad P' \rightarrow Q' \quad Q' \equiv Q}{P \rightarrow Q}$ $\qquad$ (CONTEXT) $\dfrac{P \rightarrow Q}{E\{P\} \rightarrow E\{Q\}}$

Fig. 1. Reduction: $P \rightarrow Q$

Ambient mobility is governed by the rules (ENTER) and (EXIT) of the Mobile Ambients. Communication can be local, as in Mobile Ambients, or across ambient boundaries, between parent and child. The rules for communication are different from those of [4]. The original formulation of the reduction semantics used different interaction patterns, as parent-child synchronization always involved a local prefix, as illustrated by the following example:

$$n[(x)^p P \mid p[\langle M \rangle P \mid (x)Q \mid q[\langle N \rangle^{\uparrow}]]] \tag{3}$$

the ambient n makes a downward request to read p's local value M, while the ambient q makes an upward write request to communicate its value N to its parent. With the original semantics, the input prefix of $(x)Q$ can non-deterministically synchronize with either outputs. With the new semantics, instead, the only enabled exchange in the system (3) above is the local exchange between P and Q, as synchronization requires downward and upward exchange requests to "match".

The new reductions still fit the design principles of BA, that is resource locality. An ambient can be viewed as possessing two channels: a private channel which is only available for local exchanges, and an "upward channel" which the ambient offers to its enclosing context for read and write access. There are at least two reasons in favor of the new semantics. First, it enhances the algebraic theory of the calculus, by reducing the intrinsic non-determinism of the original semantics of communication. Secondly, it enhances the typing of mobility, as mobility can be typed independently of communication (see next session). Of course, there also are tradeoffs. In fact, the new reductions require an ambient to know the names of its children in order to communicate with them. This makes it difficult to encode certain protocols, such as broadcasting a message to all the children, that were instead easily expressed with the original semantics. We leave a discussion on the relative expressive power between the two versions for future work, and focus on information flow security instead.

2.3 Static semantics

The structure of types for BA is similar to that of companion type systems for the MA [10,6].

Ambient Types. Like Mobile Ambients, Boxed Ambients are "places of conversation". However, Boxed Ambients allow more than just one "topic" of conversation: in particular, the type of an ambient shows the topic of its upward conversations, but the values it exchanges locally and with its children may have different types. More precisely, $\mathsf{amb}[E]$ is the type of all ambients whose channel for external communication carries values of type E.

Process Types. The types of processes are defined as two-place constructors $[E, F]$ that trace the types of the local (E) and upward (F) exchanges that processes with this type may have.

Capability Types. All capabilities are assigned a type constant, noted cap. This is possible, and sound, because the new semantics of communication disentangles the local exchanges of an ambient from the upward accesses attempted by any nested sub-ambients. As a consequence, ambient mobility in the new calculus is not constrained by the type of values exchanged within ambients, and is thus orthogonal to communication. Thus, the *moded types* we studied in [4] are not needed here, as ambient mobility has no constraint. To exemplify, consider the following process:

$$n[\,(x{:}W)R \mid \langle M\rangle \mid (x{:}W_1)^p P \mid (y{:}W_2)^q Q \mid p[\langle N_1\rangle^{\uparrow}] \mid q[\langle N_2\rangle^{\uparrow}]\,]$$

The process above can be safely typed with any process type, provided that (i) M, N_1, N_2 have types, respectively, W, W_1, W_2, (ii) p and q have type $\mathsf{amb}[W_1]$ and $\mathsf{amb}[W_2]$, (iii) P, Q and R have type $[W, E]$ where E is an exchange type such that $n{:}\mathsf{amb}[E]$. In particular, in the process above there is no risk of type confusion between the three exchanged values M, N_1, N_2 since read requests from children are distinct, and they do not interfere with local communication.

The typing rules are summarized in Figure 2. The system satisfies the following fundamental property:

Proposition 2.1 (Subject Reduction) *If* $\Gamma \vdash P : T$ *and* $P \rightarrow Q$ *then* $\Gamma \vdash Q : T$.

Proof. Follows as a corollary of Proposition 3.2. □

3 A Type System for Secure Information Flow

In this section we enrich the type system of BA so as to provide static safeguards against insecure flow of information in the evolution of well-types processes.

We presuppose a complete lattice of security levels $(\Sigma, \leq)$, and let ρ, σ, δ range over security levels. We then partition the elements of this lattice into two classes, "high" and "low", as formalized in the following definition.

Definition 3.1 [Low and High levels] Let $(\Sigma, \leq)$ be a complete lattice of security levels. A *security classification* is a partition of Σ into two non-empty sets L and H, with L downward closed. Based on this classification, we then define the following order: $\rho \preceq \sigma \triangleq (\rho \in \mathsf{L} \vee \sigma \in \mathsf{H})$

3.1 Types and Judgments

The types E of exchanges, and the types of processes are defined as in Section 2. The types of expressions are redefined as follows:

Expression Types	$W ::=$	$\mathsf{amb}[\sigma, E]$	ambient
	$\mid$	$\mathsf{ucap}[\sigma]$	unsafe capability
	$\mid$	$\mathsf{scap}[\sigma]$	safe capability

Each ambient type is annotated with a security level that defines the clearance of the ambient names with that type. Capability types also have an associated security level, and are partitioned into safe and unsafe. In particular, ucap is the type of dangerous capabilities, those that are potential sources of flow of information: the typing rules will ensure that such capabilities may only be exercised within high-level ambients. Capability types are also annotated with a security level: while the annotation of ambient types is used to *assign* a security level to an ambient, the annotations of capability types are used to *record* the security of the actions performed by a process. The intuition is that in $\mathsf{cap}[\sigma]$ (with $\mathsf{cap} \in \{\mathsf{ucap}, \mathsf{scap}\}$), σ is the greatest lower bound of (the security levels of) the capabilities on a path. This is formalized by the following "cap-type" composition:

▷ $\mathsf{scap}[\sigma] \cdot \mathsf{scap}[\delta] = \mathsf{scap}[\sigma \sqcap \delta]$
▷ $\mathsf{ucap}[\sigma] \cdot \mathsf{ucap}[\delta] = \mathsf{scap}[\sigma] \cdot \mathsf{ucap}[\delta] = \mathsf{ucap}[\delta] \cdot \mathsf{scap}[\sigma] = \mathsf{ucap}[\sigma \sqcap \delta]$

where $\sqcap$ is relative to the order $\preceq$ introduced in Definition 3.1.

(ENV EMPTY)
$$\frac{}{\varnothing \vdash \diamond}$$

(ENV NAME)
$$\frac{\Gamma \vdash \diamond \quad a \notin Dom(\Gamma)}{\Gamma, a : W \vdash \diamond}$$

(PROJECTION)
$$\frac{\Gamma, a : W, \Gamma' \vdash \diamond}{\Gamma, a : W, \Gamma' \vdash a : W}$$

(SUB PROC)
$$\frac{E \in \{\mathsf{shh}, E'\}, \quad F \in \{\mathsf{shh}, F'\}}{[E, F] \leqslant [E', F']}$$

(SUBSUMPTION)
$$\frac{\Gamma \vdash P : T \quad T \leqslant T'}{\Gamma \vdash P : T'}$$

(IN)
$$\frac{\Gamma \vdash M : \mathsf{amb}[E]}{\Gamma \vdash \mathsf{in}\ M : \mathsf{cap}}$$

(OUT)
$$\frac{\Gamma \vdash M : \mathsf{amb}[E]}{\Gamma \vdash \mathsf{out}\ M : \mathsf{cap}}$$

(PATH)
$$\frac{\Gamma \vdash M_1 : \mathsf{cap} \quad \Gamma \vdash M_2 : \mathsf{cap}}{\Gamma \vdash M_1.M_2 : \mathsf{cap}}$$

(PREFIX)
$$\frac{\Gamma \vdash M : \mathsf{cap} \quad \Gamma \vdash P : [E, F]}{\Gamma \vdash M.P : [E, F]}$$

(PAR)
$$\frac{\Gamma \vdash P : [E, F] \quad \Gamma \vdash Q : [E, F]}{\Gamma \vdash P \mid Q : [E, F]}$$

(NEW)
$$\frac{\Gamma, n : \mathsf{amb}[G] \vdash P : [E, F]}{\Gamma \vdash (\boldsymbol{\nu} n{:}\mathsf{amb}[G])P : [E, F]}$$

(AMB)
$$\frac{\Gamma \vdash M : \mathsf{amb}[E] \quad \Gamma \vdash P : [F, E]}{\Gamma \vdash M[P] : [\mathsf{shh}, \mathsf{shh}]}$$

(DEAD)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : [\mathsf{shh}, \mathsf{shh}]}$$

(REPL)
$$\frac{\Gamma \vdash P : [E, F]}{\Gamma \vdash !P : [E, F]}$$

(INPUT)
$$\frac{\Gamma, \tilde{x}{:}\tilde{W} \vdash P : [\tilde{W}, E]}{\Gamma \vdash (\tilde{x}{:}\tilde{W})P : [\tilde{W}, E]}$$

(OUTPUT)
$$\frac{\Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash P : [\tilde{W}, E]}{\Gamma \vdash \langle \tilde{M} \rangle P : [\tilde{W}, E]}$$

(INPUT $\uparrow$)
$$\frac{\Gamma, \tilde{x}{:}\tilde{W} \vdash P : [E, \tilde{W}]}{\Gamma \vdash (\tilde{x}{:}\tilde{W})^{\uparrow} P : [E, \tilde{W}]}$$

(OUTPUT $\uparrow$)
$$\frac{\Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash P : [E, \tilde{W}]}{\Gamma \vdash \langle \tilde{M} \rangle^{\uparrow} P : [E, \tilde{W}]}$$

(INPUT M)
$$\frac{\Gamma \vdash M : \mathsf{amb}[\tilde{W}] \quad \Gamma, \tilde{x}{:}\tilde{W} \vdash P : [G, H]}{\Gamma \vdash (\tilde{x}{:}\tilde{W})^{M} P : [G, H]}$$

(OUTPUT N)
$$\frac{\Gamma \vdash N : \mathsf{amb}[\tilde{W}] \quad \Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash P : [G, H]}{\Gamma \vdash \langle \tilde{M} \rangle^{N} P : [G, H]}$$

Fig. 2. Type system

The next step is to determine the security clearance of the values that are exchanged in a process communication. This is formalized by the following *level* function α : *Exchange Types* $\rightarrow$ *Security Levels*, where $\mathsf{cap} \in \{\mathsf{ucap}, \mathsf{scap}\}$, and $\perp$

is the bottom element in the lattice of security levels.

$$\begin{aligned}\alpha(\mathsf{amb}[\sigma, E]) &= \sigma\\ \alpha(W_1 \times \cdots \times W_n) &= \sqcup\{\alpha(W_1), \ldots, \alpha(W_n)\}\\ \alpha(\mathsf{shh}) &= \alpha(\mathsf{cap}[\sigma]) = \bot\end{aligned}$$

As we anticipated in the Introduction, we are thus stipulating that capabilities should always be considered "low-level" values, as passing a capability does not disclose the name occurring in the capability. Notice, furthermore, that the type of a capability does trace the level of the target ambient: this information is needed to detect flows of information resulting from exercising (as opposed to exchanging) the capability in question.

The type system is defined in terms of the following classes of judgments.

$\Gamma \vdash \diamond$	Well-formed Type Environment
$\Gamma \vdash E$	Well-formed Exchange Type
$\Gamma \vdash_{\sigma} [E, F]$	Well-formed Process Type at level σ
$\Gamma \vdash M : W$	Well-typed Expression
$\Gamma \vdash_{(\sigma,\rho)} P : [E, F]$	Well-typed Process

The judgments for well-formed (exchange and process) types are functional to enforce a safe flow of data along the (anonymous) communication channels inside and across ambient boundaries. In the judgment for well-typed processes, we use two annotations on the turnstile, with the following intended meaning: σ is the clearance of the ambient enclosing P (if any), while ρ is the lower-bound on the clearance of the actions encountered so far, and it helps define the clearance at which P should type-check. To understand the rationale of the typing rules, consider the following examples (as usual, ℓ denotes a low-level, while h and k are high-level).

▷ the process $\ell[(x)^h\langle M\rangle^{\uparrow}]$ is not safe, because the observable upward exchange of M is enabled as a result of ℓ exchanging a value with the high-level subambient h. Observing an upward communication on ℓ may thus reveal the presence of the high level ambient h within ℓ. The very same reasoning shows that, instead, $l[(x)^h\mathbf{0} \mid \langle M\rangle^{\uparrow}]$ is a secure process.

Flows of information may arise from subtler combinations of high-level and low-level actions. In particular, such actions need not occur sequentially as suggested by the example above. An implicit flow of information, may also arise as a result of running two parallel threads:

▷ the process $\ell[(x)^h\langle N\rangle P \mid (y)\langle M\rangle^{\uparrow}]$ is not secure because the local exchange "links" the two threads, thus determining a causal dependency, and hence an implicit flow of information.

Both the previous examples, show that "secure" processes should satisfy a very basic invariant, namely that "actions" following a high-level synchronization (like $(x)^h$) must not be available for further low-level-context interactions. This explains the role of ρ in the typing judgement of processes. When prefixing a process P with an "action" (where action means capability, communication, and top level presence of an ambient), that action should have clearance not lower than ρ. In other words, ρ should be non-decreasing as a well-typed process progresses. However, this condition is not sufficient by itself.

▷ consider the process $P = h[\ell[(x)^k \mathsf{out}\ h.\mathbf{0}]]$, where a low-level ambient ℓ first reads from a high-level ambient k, and then exits from the high-level location h. In this case, the very presence, at top level, of the ambient ℓ represents a public (low-level) information that depends in a private (high-level) one. This is a problem, as the ability to test the presence of ℓ at top level may, implicitly, reveal the presence of h to any low-level observer. To see the problem, and phrase it in terms of non-interference, we may encode the observer as the context: $\mathbf{C}() = \ell_1[\mathsf{in}\ \ell.\mathsf{out}\ \ell.\langle N\rangle^{\uparrow}] \mid ()$. Now, taking $H = k[\mathsf{in}\ h.\mathsf{in}\ \ell.\langle M\rangle^{\uparrow}\mathbf{0}]$, a routine check verifies that the context distinguishes P from $P \mid H$.

This last example shows that low-level ambients exiting high-level locations may potentially disclose secret information about that high-level location. This suggests (i) that the out capability should be deemed unsafe when the target ambient is high-level, and (ii) that only high-level ambients should be allowed to exercise such capability.

3.2 Typing Rules

Environment and Type Formation

As we anticipated, the rules for well-formed types provide safeguards against explicit flows, in that they guarantee that σ-level values only circulate over channels (or ambients) with higher clearance. This is obtained by requiring that the clearance σ of an ambient a be an upper bound on the clearance of its upward exchanges (rule TYPE AMB) and on the exchanges performed by the processes it contains (rule TYPE PROC).

(ENV EMPTY)
$$\frac{}{\varnothing \vdash \diamond}$$

(ENV NAME)
$$\frac{\Gamma \vdash T \quad a \notin Dom(\Gamma)}{\Gamma, a : W \vdash \diamond}$$

(TYPE SHH)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{shh}}$$

(TYPE CAP)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathsf{cap}[\sigma]}$$

(TYPE AMB)
$$\frac{\Gamma \vdash E \quad \alpha(E) \preceq \sigma}{\Gamma \vdash \mathsf{amb}[\sigma, E]}$$

(TYPE PROC)
$$\frac{\Gamma \vdash E_i \quad \alpha(E_i) \preceq \sigma \quad i = 1,2}{\Gamma \vdash_\sigma [E_1, E_2]}$$

Subtyping and Subsumption

The relation of subtyping coincides with the one defined for the system of Section 2. The rule of subsumption requires the target type to be well-formed to enable type promotion.

$$\frac{E \in \{\mathsf{shh}, E'\}, \quad F \in \{\mathsf{shh}, F'\}}{[E,F] \leqslant [E',F']} \; (\textsc{Sub Proc})$$

$$\frac{\Gamma \vdash_{(\sigma,\rho)} P : [E,F] \quad [E,F] \leqslant [E',F'] \quad \Gamma \vdash_{\sigma} [E',F']}{\Gamma \vdash_{(\sigma,\rho)} P : [E',F']} \; (\textsc{Subsumption})$$

Expressions

As suggested by the last example of § 3.1, a capability $\mathsf{out}\ n$ should be considered unsafe if n is high-level. On the other hand, an in capability may safely be exercised by any ambient (a low-level ambient ℓ entering a high-level ambient h may create a flow of information, but only if ℓ were allowed to eventually exit h).

$$\frac{\Gamma, a : W, \Gamma' \vdash \diamond}{\Gamma, a : W, \Gamma' \vdash a : W} \; (\textsc{Projection}) \qquad \frac{\Gamma \vdash M_1 : \mathsf{cap}[\sigma_1] \quad \Gamma \vdash M_2 : \mathsf{cap}[\sigma_2] \quad (\mathsf{cap} \in \{\mathsf{scap}, \mathsf{ucap}\})}{\Gamma \vdash M_1.M_2 : \mathsf{cap}[\sigma_1] \cdot \mathsf{cap}[\sigma_2]} \; (\textsc{path})$$

$$\frac{\Gamma \vdash M : \mathsf{amb}[\sigma, E]}{\Gamma \vdash \mathsf{in}\ M : \mathsf{scap}[\sigma]} \; (\textsc{In}) \qquad \frac{\Gamma \vdash M : \mathsf{amb}[\sigma, E] \quad \sigma \notin \mathsf{H}}{\Gamma \vdash \mathsf{out}\ M : \mathsf{scap}[\sigma]} \; (\textsc{Safe-Out}) \qquad \frac{\Gamma \vdash M : \mathsf{amb}[\sigma, E] \quad \sigma \in \mathsf{H}}{\Gamma \vdash \mathsf{out}\ M : \mathsf{ucap}[\sigma]} \; (\textsc{Unsafe-Out})$$

Processes

For the rules that follow, we define $\mathsf{Safe}(\sigma, \rho, \delta) \triangleq (\sigma \in \mathsf{H}) \vee (\rho \preceq \delta)$: intuitively, a process P is safe either (i) if it is contained within an high level ambient, or (ii) if the clearances of the 'actions' performed by P do not decrease as P progresses.

$$\frac{\Gamma \vdash M : \mathsf{scap}[\delta] \quad \Gamma \vdash_{(\sigma,\rho)} P : [E,F] \quad \mathsf{Safe}(\sigma,\rho,\delta)}{\Gamma \vdash_{(\sigma,\rho)} M.P : [E,F]} \; (\textsc{safe-prefix})$$

$$\frac{\Gamma \vdash M : \mathsf{ucap}[\delta] \quad \Gamma \vdash_{(\sigma,\rho)} P : [E,F] \quad (\sigma \in \mathsf{H})}{\Gamma \vdash_{(\sigma,\rho)} M.P : [E,F]} \; (\textsc{unsafe-prefix})$$

Safe prefixes are lower-bounded by ρ, following the previous intuition. As an example, the process $(x)^h \mathsf{out}\ \ell$ is well-typed only at level $\sigma \in \mathsf{H}$, as it represents a low-level action that depends from (as it follows) a high level one. Instead, unsafe prefixes may only be exercised within high-level ambients: this prevents low-level ambients from escaping from high-level contexts. Notice that mobility does not affect the lower bound ρ: this is safe and leaves a certain freedom to move (e.g. the path in $h.\mathsf{in}\ l$ can be executed also at level $\sigma \in \mathsf{L}$).

The following four rules are standard, and should be self-explanatory.

(PAR)
$$\frac{\Gamma \vdash_{(\sigma,\rho)} P : [E,F] \quad \Gamma \vdash_{(\sigma,\rho)} Q : [E,F]}{\Gamma \vdash_{(\sigma,\rho)} P \mid Q : [E,F]}$$

(DEAD)
$$\frac{\Gamma \vdash_{\sigma} [E,F]}{\Gamma \vdash_{(\sigma,\rho)} \mathbf{0} : [E,F]}$$

(NEW)
$$\frac{\Gamma, n : \mathsf{amb}[\tau,G] \vdash_{(\sigma,\rho)} P : [E,F]}{\Gamma \vdash_{(\sigma,\rho)} (\boldsymbol{\nu} n : \mathsf{amb}[\tau,G])P : [E,F]}$$

(REPL)
$$\frac{\Gamma \vdash_{(\sigma,\rho)} P : [E,F]}{\Gamma \vdash_{(\sigma,\rho)} !P : [E,F]}$$

The (AMB) rule implements the idea that an ambient is viewed as an "action". That is why the rule needs the hypothesis $\mathsf{Safe}(\sigma,\rho,\delta)$ as in rule (SAFE-PREFIX). Furthermore, the process enclosed in M is typed at level δ (the clearance of M) and with ρ initially set to the bottom security level.

(AMB)
$$\frac{\Gamma \vdash M : \mathsf{amb}[\delta,E] \quad \Gamma \vdash_{(\delta,\bot)} P : [F,E] \quad \Gamma \vdash_{\sigma} [G,H] \quad \mathsf{Safe}(\sigma,\rho,\delta)}{\Gamma \vdash_{(\sigma,\rho)} M[P] : [G,H]}$$

We finally come to the rules for communication, which test the predicate Safe in ways similar to the rules for prefixes. In addition, exchanging a value affects the lower bound ρ in the typing of the continuation process P. Thus, when typed at level $\sigma \in \mathsf{L}$, a process may safely communicate with a high level subambient, provided that all the subsequent actions are high-level. Thus, for instance, the processes $\ell[(x)^h\langle x\rangle^{h_1}]$ and $l[\langle \ell_1\rangle\langle M\rangle^h]$ are well typed, while $\ell[\langle M\rangle^h\langle \ell_1\rangle]$ is not.

(INPUT)
$$\frac{\Gamma, \tilde{x}{:}\tilde{W} \vdash_{(\sigma,\alpha(\tilde{W}))} P : [\tilde{W},E] \quad \mathsf{Safe}(\sigma,\rho,\alpha(W_i))}{\Gamma \vdash_{(\sigma,\rho)} (\tilde{x}{:}\tilde{W})P : [\tilde{W},E]}$$

(OUTPUT)
$$\frac{\Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash_{(\sigma,\alpha(\tilde{W}))} P : [\tilde{W},E] \quad \mathsf{Safe}(\sigma,\rho,\alpha(W_i))}{\Gamma \vdash_{(\sigma,\rho)} \langle \tilde{M}\rangle P : [\tilde{W},E]}$$

(INPUT M)
$$\frac{\Gamma \vdash M : \mathsf{amb}[\delta,\tilde{W}] \quad \Gamma, \tilde{x}{:}\tilde{W} \vdash_{(\sigma,\delta)} P : [E,F] \quad \mathsf{Safe}(\sigma,\rho,\delta)}{\Gamma \vdash_{(\sigma,\rho)} (\tilde{x}{:}\tilde{W})^M P : [E,F]}$$

(OUTPUT N)

$$\frac{\Gamma \vdash N : \mathsf{amb}[\delta, \tilde{W}] \quad \Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash_{(\sigma,\delta)} P : [E, F] \quad \mathsf{Safe}(\sigma, \rho, \delta)}{\Gamma \vdash_{(\sigma,\rho)} \langle \tilde{M} \rangle^N P : [E, F]}$$

(INPUT $\uparrow$)

$$\frac{\Gamma, \tilde{x}{:}\tilde{W} \vdash_{(\sigma,\alpha(\tilde{W}))} P : [E, \tilde{W}] \quad \mathsf{Safe}(\sigma, \rho, \alpha(W_i))}{\Gamma \vdash_{(\sigma,\rho)} (\tilde{x}{:}\tilde{W})^{\uparrow} P : [E, \tilde{W}]}$$

(OUTPUT $\uparrow$)

$$\frac{\Gamma \vdash \tilde{M} : \tilde{W} \quad \Gamma \vdash_{(\sigma,\alpha(\tilde{W}))} P : [E, \tilde{W}] \quad \mathsf{Safe}(\sigma, \rho, \alpha(W_i))}{\Gamma \vdash_{(\sigma,\rho)} \langle \tilde{M} \rangle^{\uparrow} P : [E, \tilde{W}]}$$

As usual, the correctness of the type system is guaranteed by the subject reduction property.

Proposition 3.2 (Subject Reduction)
If $\Gamma \vdash_{(\sigma,\rho)} P : [E, F]$ *and* $P \rightarrow Q$ *then* $\Gamma \vdash_{(\sigma,\rho)} Q : [E, F]$.

It is rather straightforward to show that the type system detects, and prevents, all unsafe forms of explicit flow, in the sense of property ($\star$) in the Introduction (cf. page 3). More interestingly, we can show that unsafe implicit flows are also detected: this is the topic of the next section.

4 Non-interference

We start introducing the notion of 'high-level sources', in terms of which we then state our NDC-based definition of non-interference.

Definition 4.1 [High-level Sources] A process P is a *high-level source* if and only if (i) $\Gamma \vdash_{(\sigma,\rho)} P : T$, for all security levels σ and ρ with $\rho \in \mathsf{H}$, and (ii) if P is of the form $M.P'$ then $\Gamma \vdash M : \mathsf{cap}[\delta]$ with $\delta \in \mathsf{H}$.

Accordingly, high-level sources are well-typed processes that may only engage 'high' top-level interactions with any context in which they are inserted. This is true of processes in prefixed form by virtue of condition (ii). In addition, an inspection of the typing rules verifies the following properties of any high level source P. First, all the top-level value exchanges with P must be high-level, and so must be all the top-level ambient occurrences in P. Secondly, the well-typedness condition ensures that no low-level ambient may escape its enclosing high-level contexts.

Notation: We henceforth write $\Gamma \Vdash P : T$ to indicate that P is a high-level source in Γ. Also, we write $\Gamma \vdash P : [E, F]$ and $\Gamma \vdash P : ok$ when σ, ρ and/or $[E, F]$ are not relevant.

4.1 Typed Equivalence

Next, we introduce a *typed* notion of process equivalence. The equivalence is typed as we compare only processes with the same types, and inserted in contexts that respect their typing. We formalize these notions below following [22].

A *context* $\mathbf{C}()$ is a process term with just one hole $()$. We denote with $\mathbf{C}(P)$ the process resulting from replacing the hole with P in $\mathbf{C}()$. Note that variables and names that are free in P may become bound in $\mathbf{C}(P)$. Thus we do not identify contexts up to renaming of bound variables and names.

Definition 4.2 [Γ/Δ Context] Let Γ and Δ be type environments and T a process type. $\mathbf{C}()$ is a $(\Gamma/\Delta, T)$-context if $\Gamma \vdash_{(\sigma,\rho)} \mathbf{C}() : ok$ with $\sigma \in \mathsf{L}$ is derivable in the type system of Section 3 enriched with the a rule that derives $\Theta \vdash () : T$ for all Θ extending Δ.

Intuitively, a $(\Gamma/\Delta, T)$-context is a context whose hole, of type T, is in the scope of the binders recorded in Δ, and whose free names and variables are contained in Γ. Furthermore, the context $\mathbf{C}()$ must be typed at low level, that is the clearance of external observers.

Definition 4.3 [Barbs] Define $P \downarrow_n \triangleq\ P \equiv (\nu \boldsymbol{m})(n[\langle M\rangle^{\uparrow} P' \mid Q'] \mid Q'')\ n \notin \{\boldsymbol{m}\}$. A process P exhibits the name n, written $P \Downarrow_n$ iff there exists Q such that $P \Longrightarrow Q$ and $Q \downarrow_n$, where $\Longrightarrow$ is the reflexive and transitive closure of $\rightarrow$.

Now we can define our notion of 'low' typed equivalence, relative to an underlying security classification into 'low' and 'high' levels. Based on that we then have our definition of the non-interference.

Definition 4.4 [Typed observational equivalence and Non-interference] Assume $\Delta \vdash P : T$ and $\Delta \vdash Q : T$. The two processes are equivalent in Δ, written $\Delta \triangleright P \cong_{\mathsf{L}} Q$ if and only if for all $(\Gamma/\Delta, T)$-context $\mathbf{C}()$ with $\mathbf{C}(P)$ and $\mathbf{C}(Q)$ closed, for all n with $\alpha(\Gamma(n)) \in \mathsf{L}$, $\mathbf{C}(P) \Downarrow_n \Leftrightarrow\ \mathbf{C}(Q) \Downarrow_n$

Definition 4.5 [Non-interference] Let Σ be a security lattice and P a process. Given a security classification of Σ such that $\Delta \vdash P : T$, P is *secure* for that classification iff $\Delta \triangleright P \cong_{\mathsf{L}} P \mid H$ for all H such that $\Delta \Vdash H : T$. P is *interference-free* if it secure for all security classifications of Σ.

We conclude with the main result, a theorem that states that the type system guarantees non-interference for well-typed processes.

Theorem 4.6 (Non-interference)
Given any security classification, if $\Delta \vdash P : T$ and $\Delta \Vdash H : T$, then $\Delta \triangleright P \cong_{\mathsf{L}} P \mid H$.

Notice that the theorem is stated, and proved, only in reference to well-typed contexts. Accordingly, the non-interference analysis it addresses corresponds to verifying the 'internal' security of a system rather than its security with respect to external attackers.

The non-interference proof draws on the technical tools developed by Cardelli and Gordon for Mobile Ambients [9], adapting them to our Boxed Ambients. The non-interference result derives from a lemma that shows that high-level sources are low-level equivalent to the inactive process. More precisely, we show that for every context $\mathbf{C}()$ and high-level source H if $\mathbf{C}(H)$ is well-typed then it is indistinguishable at low level from $\mathbf{C}(\mathbf{0})$. Proving this result requires a characterization of all the possible interactions between a process and the surrounding context.

4.2 Discussion

The type system we have defined to derive the non-interference proof is admittedly somewhat restrictive. While this is unfortunate, the discipline we impose on interactions between high and low-level processes has effects comparable to those found in existing type systems for secure information-flow in simpler process calculi [16], and multi-threaded languages [26,24,1] (cf. Section 5 for a detailed comparison).

Also, even though well-typed processes are constrained in the actions they may perform, the type system still allows non-trivial forms of interaction between high and low levels, both in terms of mobility, and of value exchanges. Figure 3 shows the legal flow of information for a well-typed composition of the two processes P and H, when H is a high-level source.

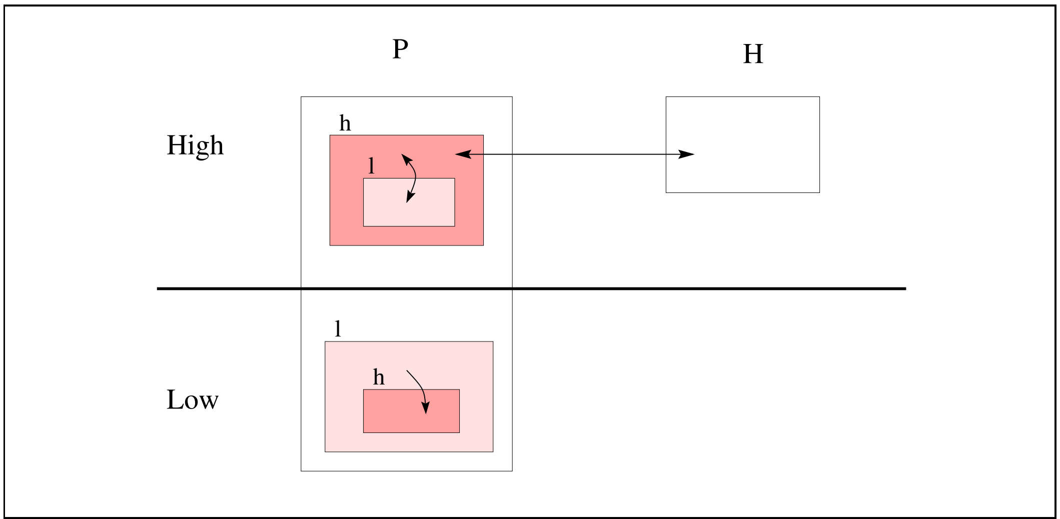

Fig. 3. Flows of information of $P \mid H$

In particular, the flows enabled by the type system are (i) those from H to the high sub-processes of P (and vice versa), and (ii) those from the high-level components of P to those low components of P that are not observable since they are shielded by high-level ambients. A low-level observer may thus observe only flows of information between low-level components of P and low level components of the surrounding context.

Also note that high-level information can be freely exchanged between the high and low level processes of P, as long as the latter are nested within high-level ambients. This is because the type system ensures that these low sub-processes are confined within high-level ambients.

In the calculus of Mobile Ambients, a similar property would be harder to enforce. This is because the open capability represents an *objective* action that the context may impose on a process. To see the consequences of that, note that to prove our non-interference theorem in MA, a process P should be checked against all high level processes that appear in parallel with P, in particular against the high-level processes $\mathsf{open}\ h.\mathbf{0}$, for all high-level names h. This implies that processes of the form $h[P]$, with P low-level, should be rejected by the type system as not secure. To motivate, consider the process $P = h[\ell[P']]$, for *any* P', and with h and ℓ high and low-level names, respectively. This process is not interference-free, as the context $\mathbf{C}() = \ell_1[\mathsf{in}\ \ell.\mathsf{out}\ \ell.\langle M\rangle^{\uparrow}] \mid ()$ may distinguish between P and $P \mid \mathsf{open}\ h.\mathbf{0}$. A similar reasoning applies to the processes $h[\mathsf{in}\ \ell.P \mid Q]$ and $h[\mathsf{out}\ l.P \mid Q]$, and in general to any process $h[P']$ where P' is a low-level process. All such processes are instead well-typed, and interference-free in our calculus, under the additional assumption that the low-level components of P' do not attempt to escape outside h.

As a further remark, we note that the proof of non-interference would not go through in the presence of finer equivalence relations such as barbed congruence, bisimulation or must testing. To see the problem with barbed congruence, consider defining $\approx_{\mathsf{L}}$ as the barbed congruence relation induced by our observability predicate $P \downarrow_n$. Then, take the processes $P = \ell[\langle M\rangle^{\uparrow} \mid \mathsf{in}\ h.\mathbf{0}]$ and the high-level process $H = h[\,]$. Now take the context $\mathbf{C}() = ()$, and observe that $\mathbf{C}(P \mid H) \rightarrow R = h[\ell[\langle M\rangle^{\uparrow}]]$, while there exists no process R' such that $\mathbf{C}(P) \rightarrow R'$ and $R \approx_{\mathsf{L}} R'$.

5 Related Work

Volpano and Smith [26,24,25], and recently Boudol and Castellani [1] study type-based techniques to enforce non-interference in multi-threaded imperative languages. In their approach explicit flow is prevented by imposing constraints on variable assignments, while additional restrictions on conditional commands and while-loops rule out implicit flow. In [1] the authors point out that introducing parallelism may cause new problems, since information flow may be “disguised as control flow”, and a program may observe (and be influenced by) the behavior of other concurrent components in the course of their execution. The problem is solved in [25,1] by relying on a form of asynchrony, whereby consulting the value of a high-level variable must not be followed by an assignment to a low variable. In BA we have a similar problem even though in a different setting, and our solution follows the same rationale, by imposing a non-decreasing clearance on the sequence of ‘actions’ performed by a process.

More directly related to ours are the type systems for π-calculus by Honda et al [17] and for the *security π calculus* by Hennessy and Riely [16]. In [17], the authors propose to use the informal principle of causal dependency to understand safety of information flow in various programming language, and develop a type system based on a behavioral notion of types to capture causality of actions. Our approach is similar, as it also draws on the principle of causal dependency, but our framework appears to be more complex, as in BA processes may interact both via communications and mobility.

In [16], security levels are attached to processes and to capabilities for reading/writing to channels, and a 'no read-up/no write-down' security policy is enforced by typing. To prove non-interference, further restrictions must be imposed, namely high-level processes must not evolve in low-level ones and the calculus must be asynchronous. Under these hypotheses, the authors show that well-typed asynchronous processes are interference free, where non-interference is defined in a way similar to ours, based on *may test* equivalence. Our type system enforces similar restrictions on the value exchanges between high and low processes, and corresponding restrictions on mobility. Unlike [16], our result holds true for the synchronous case as well. In [15], Hennessy has developed an enhanced type system for teh security π calculus for which non-interference can be proved also with respect to *must* test equivalence.

In [23] Sewell and Vitek introduce *box*-π, a process calculus that provides mechanisms for composing (partially trusted) software components and for enforcing information flow security policies. Their approach is based on a colored semantics, which annotates output processes with the sets of principals that have affected them (the processes) in the past; then the security properties are stated in terms of a colored lts. Finally, they introduce a type system that statically captures causal flows. As such, the characterization of information flow security is based on a causal model, rather then on non-interference as in our approach. Further important differences are the asynchronous semantics of box-π (as opposed to the synchronous semantics of BA) and our treatment of mobility and nested topology. A more in-depth comparison between the two approaches deserves to be made.

No type-based study of non-interference appears to have been conducted on ambient-based calculi. A number of papers have instead dealt with other aspects of security. Cardelli et al. present a type system for Mobile Ambients [7] based on the notion of group names, that statically prevents unwanted propagation of names. The typing system by Levi and Sangiorgi [18] for Safe Ambients provides finer control over ambient interactions and prevents 'grave interferences'. Dezani and Salvo, in [12], develop a type system for Mobile Ambients in which ambient types are associated with security levels in ways similar to ours, and security checks are over opening and moves.

Other approaches based on type systems [3] and control-flow analyses have also been applied [21,20] to analyze different security properties of (various dialects of) mobile ambients. In particular Braghin et al. [2] study 'explicit' information flow security in the scenario of pure Mobile Ambients by defining a control-flow analysis to detect security breaches arising as confidential data moving outside any voundary protection.

6 Conclusions

We have studied information flow security in the calculus of Boxed Ambients. We have developed a notion of non-interference based on a typed equivalence induced by "low level observations", and presented a sound type system whose well-typed processes are guaranteed to be interference-free. To our knowledge, no such study has been conducted in the existing literature.

Plans of future work include the development of refined type systems capable of capturing stronger non interference properties based on stricter equivalences, and of type and effect systems allowing more flexibility in the typing of value exchanges and mobility. Also, it would be desirable to extend the non-interference proof to the case of partially-typed systems.

References

[1] G. Boudol and I. Castellani. Non-interference for concurrent programs. In *Proceedings of ICALP'2001*, number 2076 in Lecture Notes in Computer Science, pages 328–395. Springer, 2001.

[2] Chiara Braghin, Agostino Cortesi, and Riccardo Focardi. Control Flow Analysis of Mobile Ambients with Security Boundaries. In B. Jacobs and A. Rensink, editors, *Proc. of Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'02)*, pages 197–212. Kluwer Academic Publisher, 2002.

[3] M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages*, pages 222–235, London, 2001. ACM Press.

[4] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *TACS 2001 (4th. International Symposium on Theoretical Aspects of Comuter Science)*, number 2215 in Lecture Notes in Computer Science, pages 38–63, Sendai, Japan, 2001. Springer.

[5] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *CONCUR 2001 (12th. International Conference on Concurrency Theory)*, number 2154 in Lecture Notes in Computer Science, pages 102–120, Aalborg, Danemark, 2001. Springer.

[6] L. Cardelli, G. Ghelli, and A. Gordon. Mobility types for mobile ambients. In *Proceedings of ICALP'99*, number 1644 in Lecture Notes in Computer Science, pages 230–239. Springer, 1999.

[7] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science, pages 333–347. Springer, August 2000.

[8] L. Cardelli and A. Gordon. Mobile ambients. In *Proceedings of F0SSaCS'98*, number 1378 in Lecture Notes in Computer Science, pages 140–155. Springer, 1998.

[9] L. Cardelli and A. Gordon. Equational properties for mobile ambients. In *Proceedings FoSSaCS'99*. Springer LNCS. Full version available as Microsoft Research Technical Report MSR-TR-99-11, 1999.

[10] L. Cardelli and A. Gordon. Types for mobile ambients. In *Proceedings of POPL '99*, pages 79–92. ACM Press, 1999.

[11] G. Castagna, G. Ghelli, and F. Zappa. Typing mobility in the Seal Calculus. In *CONCUR 2001 (12th. International Conference on Concurrency Theory)*, number 2154 in Lecture Notes in Computer Science, pages 82–101, Aalborg, Danemark, 2001. Springer.

[12] M. Dezani-Ciancaglini and I. Salvo. Security types for safe mobile ambients. In *Proceedings of ASIAN'00*, pages 215–236. Springer, 2000.

[13] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[14] J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of Symposium on Secrecy and Privacy*, pages 11–20. IEEE Computer Society, april 1982.

[15] M. Hennessy. The security picalculus and non-interference. Technical Report CS-05-2000, University of Sussex, School of Cognitive and Computing Sciences, BRIGHTON BN1 9QH, UK, Nov. 2000.

[16] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous π-calculus (extended abstract). In *Automata, Languages and Programming, 27th International Colloquium*, volume 1853 of *Lecture Notes in Computer Science*, pages 415–427. Springer, 2000.

[17] K. Honda, V.T. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In *ESOP '00*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer, 2000.

[18] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL2000*, pages 352–364. ACM Press, 2000.

[19] R. Milner and D. Sangiorgi. Barbed bisimulation. In *ICALP'92*, number 623 in Lecture Notes in Computer Science, pages 685–695. Springer, 1992.

[20] F. Nielson and H.R. Nielson. Shape analysis for mobile ambients. In *POPL'00*, pages 135–148. ACM Press, 2000.

[21] F. Nielson, H.R. Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in mobile ambients. In *CONCUR'99*, number 1664 in Lecture Notes in Computer Science, pages 463–477. Springer, 1999.

[22] D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[23] P. Sewell and J. Vitek. Secure composition of untrusted code: Wrappers and causality types. In *13th IEEE Computer Security Foundations Workshop*, 2000. To Appear in *Journal of Computer Security*.

[24] D. Volpano and G. Smith. A type-based approach to program security. In *Proc. 7th Int'l Joint Conference on the Theory and Practice of Software Development*, number 1214 in Lecture Notes in Computer Science, pages 607–621. Springer, 1997.

[25] D. Volpano and G. Smith. Secure information flow in a multi-threaded imperative langguage. In *Proc. of POPL'98*, pages 355–364. ACM Press, 1998.

[26] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.