# Role-based Access Control for Boxed Ambients ⋆

Adriana Compagnoni [a] Elsa L. Gunter [b] Philippe Bidinger [c]

[a] *Stevens Institute of Technology*
[b] *University of Illinois, Urbana - Champaign*
[c] *VERIMAG, Grenoble, France*

*In honor of Mario Coppo, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca on the occasion of their sixtieth birthdays*

## Abstract

Our society is increasingly moving towards richer forms of information exchange where mobility of processes and devices plays a prominent role. This tendency has prompted the academic community to study the security problems arising from such mobile environments, and in particular, the security policies regulating who can access the information in question.

In this paper we describe a calculus for mobile processes and propose a mechanism for specifying access privileges based on a combination of the identity of the users seeking access, their credentials, and the location from which they seek it, within a reconfigurable nested structure.

We define $\mathbf{BACI_R}$, a boxed ambient calculus extended with a *Distributed Role-Based Access Control* mechanism where each ambient controls its own access policy. A process in $\mathbf{BACI_R}$ is associated with an owner and a set of activated roles that grant permissions for mobility and communication. The calculus includes primitives to activate and deactivate roles. The behavior of these primitives is determined by the process's owner, its current location and its currently activated roles. We consider two forms of security violations that our type system prevents: 1) attempting to move into an ambient without having the authorizing roles granting entry activated and 2) trying to use a communication port without having the roles required for access activated. We accomplish 1) and 2) by giving a static type system, an untyped transition semantics, and a typed transition semantics. We then show that a well-typed program never violates the dynamic security checks.

*Preprint submitted to Elsevier* *25 October 2007*

## 1 Introduction

The exchange of information by electronic means in a mobile environment has become commonplace, with cellphones, PDA's, and laptop computers accessing remote information and transmitting signals and data. An increasingly mobile workforce needs to be able to access corporate information while at work, from home, and on the road. This tendency has led the academic community to study the security problems arising from this ever increasing mobility.

One aspect of secure computing is the control of who gains access to which shared and sensitive computing resources. *Role-Based Access Control* (RBAC) [15,26,16] is a standardized methodology for defining security policies and for giving privileges to users, based on using *roles* as an abstraction representing a set of activities to be performed. Access is fundamentally controlled by roles. On one side, each user of a system is associated with a set of roles. On the other side, each role is associated with a set of permissions (access privileges to existing resources). Some roles may be mutually exclusive, and others may be deactivated leaving the user with only a subset of the full set of roles with which she is associated. Therefore, in simple RBAC, a user is granted an access privilege to a resource if one of her activated roles has that privilege. This factorization of access control simplifies the administration of the security policy by allowing the systems administrator to separately decide which resources a given role needs in order to successfully operate, and what roles to assign to each user. It also allows for the choice of authentication method to be handled separately. How to enrich RBAC by adding orderings and other forms of structure on the roles and the privileges is an active area of research. They all share in common the separation of concerns given by the introduction of roles. RBAC, however, is not concerned with the authentication of users. Whether the user claiming to be Bob is indeed Bob precedes the application of access control, and is beyond the scope of this work.

Mobility adds a new dimension to RBAC, since the services available to a given user also depend on the location of the user, agreements between parties, and the technology underlying the connection. For example, without roaming agreements in place, a cell-phone may be rendered useless beyond the scope of

⋆ This research was partially supported by the NSF Grant No. CCR-0220286 ITR: Secure Electronic Transactions, by US Army Research Office Grant number DAAAD19-01-1-0473, and by the US Army under contract W15QKN-05-D-0011.

*Email addresses:* `abc@cs.stevens.edu` (Adriana Compagnoni), `elsa@cs.uiuc.edu` (Elsa L. Gunter), `philippe.bidinger@imag.fr` (Philippe Bidinger).

*URLs:* `www.cs.stevens-tech.edu/~abc` (Adriana Compagnoni), `www-faculty.cs.uiuc.edu/~egunter` (Elsa L. Gunter), `www-verimag.imag.fr/~bidinger` (Philippe Bidinger).

its provider's network. Furthermore, whether a user's connection is wireless, wired, secure, or insecure also conditions the available services. For example, an administrator on an insecure wireless connection may be denied access to sensitive information. In a distributed environment the policies regulating access control may be distributed among several parties, and each principal may only have partial knowledge of the overall security policy [24,25,23]. In a mobile environment, different domains will have different access policies and when users (and potentially programs) migrate from domain to domain the access policy governing them will change with their enclosing domain.

Role-based access control is currently a popular mechanism for governing the access to databases, files, executable programs and other computational resources. In networking there is another kind of access control that is done by packet filtering. A given router may be configured to drop all SMTP or HTTP packets denying access to certain services of a domain from outside that domain. Here, there is no notion of user and role, but only IP domain and packet type. However, it can be beneficial to have a finer-grain access control that is aware of roles and network domains. Consider the following example:

> The University of Wizbrau is equipped with intelligent buildings, and students carry their laptops with them to class. While in the classroom, students have only limited Internet access and they are not allowed to use e-mail, instant messenger, or visit general websites. However, these activities are allowed when done from the student lounge instead. Since the instructor of the course needs a greater access to resources than the students, those activities locally disabled to the students are available to the instructor. For example, during a lecture, the instructor may consult her e-mail to address a question raised by a student in an e-mail message.

The restrictions placed on users in this environment need to be sensitive to both the location of the user (classroom versus lounge) as well as the role (student versus instructor). Such fine-grained control is not readily handled by either packet filtering or RBAC.

In this paper we design a formal language featuring formal notions for resource, access, computation, communication, location and mobility. The starting point of our design is a mobile ambient calculus in the style of [10], where principals and locations are modeled by ambients. This formal language further includes a type system based on roles and localized role-based access control. We further show that well-typed programs in this system do not attempt unauthorized access to resources.

## *1.1 Background on Ambient Calculi*

In Cardelli and Gordon's Mobile Ambients (MA)[11], ambients represent nested computational environments containing data and live computation. In a nutshell, ambients are administrative units forming a dynamic hierarchy, where an ambient can move up and down the hierarchy by moving into a sibling or a parent ambient. Furthermore, a mobile ambient is a communicating entity that can exchange information with parents and children. MA are capable of moving under the influence of the process they enclose and can dissolve their perimeter with an *open* operation. Mobile Ambients provide a direct characterization of computational processes as well as computational devices.

Boxed Ambients (BA) [6] evolved from MA, by removing the ability of an ambient to dissolve its boundary. In BA, an ambient is a "box" that cannot be opened. This notion of closed ambient provides a complete encapsulation of the agents they contain. To enable the communication lost by disabling the open operation, ambients are equipped with communication channels to exchange information with adjacent ambients (parent and children ambients).

Both in MA and BA, ambient mobility is commanded by processes inside the ambient. The commands for mobility are called *capabilities.* The capabilities tell an ambient to open or move inside or outside another ambient. Unrestricted mobility, however, can lead to undesired *interferences* between two concurrent processes. To address this concern, control over capabilities was first introduced in Safe Ambients [21] and later used in New Boxed Ambients (NBA) [8] in the form of *co-capabilities.* A capability can be exercised only in the presence of a matching co-capability. Hence, in order to enter an ambient using the $\mathsf{in}$ capability, that ambient must contain a matching $\overline{\mathsf{in}}$ co-capability authorizing that access; similarly for exiting using the $\mathsf{out}$ capability.

Boxed Ambients with Communication Interfaces (**BACI**) [3], introduced the notion of *local views.* In this calculus, each ambient has an associated communication *port* and a *local view.* The communication port is used for sending and receiving messages to and from other ambients, and the local view represents the communication types that are used by the processes enclosed inside the ambient. **BACI** is flexible enough to allow an ambient to communicate with different parents using different types. However, this flexibility came with the price of a rather complex syntax and some run-time type checking required to guarantee type safety. **BACIv2** [17] further enhanced communication mechanisms and mobility control by introducing multiple communication ports, access control lists, and port hiding.

Motivated by our earlier work on **BACI** [3], we define a typed boxed ambient calculus called $\mathbf{BACI_R}$ extended with a *Distributed Role-Based Access Control* mechanism where each ambient controls its own access policy. Following the style of **BACI**, our new calculus distinguishes between names of ambients and names of communication ports. Ambients are used for mobility and ports are used for communication, either locally within the main process of an ambient or between a parent and a child. This distinction is instrumental in defining our RBAC mechanism, since it provides for a finer grain in the security policy. Each kind of ambient (as determined by its name) controls its own access policy by specifying which roles a user may activate for it, and which roles are sufficient to allow another ambient to enter it. Similarly, each kind of ambient specifies for the ports it generates which roles can read from it and which roles can write to it. The idea behind grouping ambients by name is that the name should indicate the general task to be performed, and all ambients of the same name should be uniform in the way they interact with other ambients.

An ambient in $\mathbf{BACI_R}$ is associated with an owner and a set of activated roles that grant permissions for mobility and communication. The calculus includes primitives to activate and deactivate roles. The behavior of these primitives is determined by the owner of the ambient, its current location, and its currently activated roles. In order for an ambient to activate a role, the security policy has to allow the owner of the ambient to do so. Moreover, deactivating roles should not remove the roles authorizing the ambient to be in its current location.

We consider two forms of *security violations* that our type system prevents: 1) attempting to move into an ambient without having the authorizing roles granting entry activated and 2) trying to use a communication port without having the roles required for access activated. We accomplish 1) and 2) by giving a static type system in Section 3, an untyped transition semantics, and a typed transition semantics in Section 4. We then show that a well-typed program never violates the dynamic security checks.

This paper is a revision of the extended abstract [12] that appeared in the proceedings of the international symposium *Trustworthy Global Computing, 2005*. The work here differs from that in the abstract in the following ways: Firstly, the grammar of our language, and correspondingly the type system and transition rules, have been simplified by the removal of cases that were semantically equivalent to other constructs. Secondly, our discussion of the typed dynamic semantics is abbreviated here, where it was fully expanded in the earlier version. Lastly, we have added the details of how to encode the example of the student versus the instructor in the classroom in our calculus, and described how the type system prevents the unwanted network actions from taking place. Due to the nature of the special issue, we could not include technical details such as a complete definition of the operational semantics and extensive proofs of our results.

| | |
|---|---|
| *Capabilities:* | |
| $C ::= i$ | capability variable |
| $\mid \mathsf{in}\, m$ | enter |
| $\mid \mathsf{out}\, m$ | exit |
| $\mid C_1.C_2$ | path |
| *Basic types* | |
| $\tau ::= \mathsf{amb}(\rho_{in}, \sigma)$ | ambient type |
| $\mid \mathsf{cap}(\rho_{in}, \sigma)$ | capability type |
| *Messages:* | |
| $M, N ::= m$ | ambient name |
| $\mid C$ | capability |
| *Actions:* | |
| $\pi ::= C(c{:}\sigma)$ | capability |
| $\mid K(c{:}\sigma)$ | co-capability |
| $\mid \mathsf{activate}\langle \mathsf{r} \rangle$ | activate role $\mathsf{r}$ |
| $\mid \mathsf{deactivate}\langle \mathsf{r} \rangle$ | deactivate role $\mathsf{r}$ |
| $\mid (x_1, \ldots, x_k)^{\eta}$ | input |
| $\mid \langle M_1, \ldots, M_k \rangle^{\eta}$ | output |
| *Co-Capabilities:* | |
| $K ::= \overline{\mathsf{in}}$ | allow enter |
| $\mid \overline{\mathsf{out}}$ | allow exit |
| *Communication types* | |
| $\sigma ::= \mathsf{shh}$ | no exchange |
| $\mid (\rho_r, \rho_w, \tau)$ | exchange tuple |
| *Locations:* | |
| $\eta ::= \uparrow c$ | parent port $c$ |
| $\mid \downarrow c$ | child port $c$ |
| $\mid \star$ | local |
| *Processes:* | |
| $P ::= \mathbf{0}$ | nil process |
| $\mid P_1 \mid P_2$ | composition |
| $\mid \boldsymbol{\nu}(n{:}\tau)P$ | ambient name restriction |
| $\mid \boldsymbol{\nu}(c{:}\sigma)P$ | port name restriction |
| $\mid\ !P$ | replication |
| $\mid \pi.P$ | prefixing |
| $\mid m_u[P]@\rho$ | ambient |

Table 1
Syntax of $\mathbf{BACI_R}$

## 2 Syntax of $\mathbf{BACI_R}$

Based on our earlier work on **BACI** [3], we define $\mathbf{BACI_R}$, a boxed ambient calculus with a Distributed Role-Based Access Control mechanism, where the location of an ambient conditions its privileges. The intuitive idea is that to accommodate security checking an ambient is associated with its owner and with a set of roles that are currently activated. This set of roles can be changed by activation and deactivation primitives. Whether a role can be activated or deactivated depends on the location of the ambient and its owner. This control is made explicit in the type system where the type of an ambient has a set of roles authorizing the entrance of ambients. Going back to the example, the professor can send mail because she can activate the $\mathsf{faculty\_mail}$ role, while the students can only activate the $\mathsf{student\_mail}$ role, which is not enough to qualify to send mail in the classroom.

In order to define the syntax of $\mathbf{BACI_R}$ we use the following disjoint categories of identifiers:

| | | | |
|---|---|---|---|
| User Names: | $u,\ v \in \mathit{Users}$ | Ambient Names: | $n, m \in \mathit{Amb}$ |
| Roles: | $\mathsf{r} \in \mathit{Roles}$ | Capability Variables: | $i \in \mathit{CapVar}$ |
| Port Names: | $c,\ c' \in \mathcal{C}$ | Message Identifiers: | $x \in \mathit{Amb} \cup \mathit{CapVar}$ |

| | |
|---|---|
| $!P \equiv P \mid !P$ | (Struct Rep Par) |
| $\boldsymbol{\nu}(nc{:}\varphi)\boldsymbol{\nu}(nc'{:}\varphi')P \equiv \boldsymbol{\nu}(nc'{:}\varphi')\boldsymbol{\nu}(nc{:}\varphi)P$ | (Struct Res Res) |
| $\boldsymbol{\nu}(nc{:}\varphi)(P_1 \mid P_2) \equiv P_1 \mid \boldsymbol{\nu}(nc{:}\varphi)P_2,\ \text{if } nc \notin \mathtt{fn}(P) \cup \mathtt{fp}(P)$ | (Struct Res Par) |
| $\boldsymbol{\nu}(nc{:}\varphi)m_u[P]@\rho \equiv m_u[\boldsymbol{\nu}(nc{:}\varphi)P]@\rho,\ \text{if } nc \neq m$ | (Struct Res Amb) |
| $(C_1.C_2)(c{:}\sigma).P \equiv C_1(c'{:}\sigma').(C_2(c{:}\sigma).P),\ \text{where } c' \notin \mathtt{fp}(P)$ | (Struct Prefix) |

Table 2
Structural Equivalence

We assume a fixed set *Users* of users, a fixed set *Roles* of roles, and a fixed function *UserPolicy* associating each ambient, user and set of currently activated roles with a set of roles that may become activated for the given ambient. We use $nc$ to range over $\mathit{Amb} \cup \mathcal{C}$, and $\varphi$ to be either a basic type or a communication type. The syntax of $\mathbf{BACI_R}$ is presented in Table 1.

$\mathbf{BACI_R}$ may be seen as an extension of the $\pi$-calculus. *Processes* and *Actions* are the two main syntactic categories. We add to the processes of the $\pi$-calculus that of *ambients.* Further, we modify restriction to apply only to ambient names, which are distinct from port names (*a.k.a.* channel names). The $\pi$-calculus actions of input and output are modified to provide the location of the communication port (parent, child, or local). We add to these actions those for activation and deactivation of roles and those for requesting and granting permission for movement of ambients (*capabilities* and *co-capabilities*). Capabilities and co-capabilities, in addition to controlling the movement of ambients, introduce port names and provide their scope. Capabilities include paths (sequences of movement requests) to allow directions to be passed to an ambient directing it to a specific location. All the capabilities in the path except the last involve no communication and generate no port.

We introduce the usual notion of process equivalence through the structural congruence generated by alpha conversion, associativity and commutativity of parallel composition with $\mathbf{0}$ for identity, and the rules given in Table 2. The rules for replication and restriction are fairly standard. We add a rule for allowing restriction to pass through an ambient, provided that ambient is not the one whose name is being restricted. (The functions `fn` and `fp` give the free ambient names and the free port name, respectively, in a process.) The last rule tells us that to follow a path is the same as to follow it in pieces. This makes sense because ambients can only enter one other ambient at a time.

## 3 Types for Security

Attempting to enter an ambient without an authorizing role activated is a security violation. Trying to use a communication port without having activated

Ambient Name:
$$\frac{\Gamma(m) = \mathsf{amb}(\rho_{in}, \sigma)}{\Gamma \vdash m : \mathsf{amb}(\rho_{in}, \sigma)}$$

Variable:
$$\frac{\Gamma(i) = \mathsf{cap}(\rho_{in}, \sigma)}{\Gamma \vdash i : \mathsf{cap}(\rho_{in}, \sigma)}$$

Path:
$$\frac{\Gamma \vdash C_1 : \mathsf{cap}(\rho_{in}, \sigma') \quad \Gamma \vdash C_2 : \mathsf{cap}(\rho_{in}, \sigma)}{\Gamma \vdash C_1.C_2 : \mathsf{cap}(\rho_{in}, \sigma)}$$

Enter / Exit To:
$$\frac{\Gamma \vdash m : \mathsf{amb}(\rho_{in}, \sigma)}{\Gamma \vdash \mathsf{in} \,/\, \mathsf{out}\; m \; : \mathsf{cap}(\rho_{in}, \sigma)}$$

Parent/ Child Port :
$$\frac{\Gamma(c) = \sigma}{\Gamma, m \vdash \uparrow / \downarrow c : \sigma}$$

Local:
$$\frac{\Gamma \vdash m : \mathsf{amb}(\rho_{in}, (\rho_r, \rho_w, \tau))}{\Gamma, m \vdash \star : (\mathit{Roles}, \mathit{Roles}, \tau)}$$

Table 3
Typing of Ambient Names, Capabilities, Messages, and Locations

at least one of the required roles to access the port is also a security violation. In this section we define a type system such that well-typed processes can compute without committing security violations. The type of a process is a set of roles sufficient for it to compute without security violations. In particular, the type of an ambient name is the set of roles needed for mobility and communication.

The syntax of types can be found in Table 1. Basic types describe the kind of data to be communicated over a port, either ambient name or capability, together with the rules sufficient for entrance and the communication type of the associated ambient. The communication type includes the sets of roles $\rho_r$ and $\rho_w$ granting read and write access to a port. In this presentation of the calculus, we allow only ambient names and capabilities to be passed over ports, but not port names. Allowing the communication of computational types such as the integers does not affect the results here.

The type system is defined with the following judgments. We give here only a partial presentation of system.

| | | | |
|---|---|---|---|
| Capabilites | $\Gamma \vdash C : (\rho_{in}, \sigma)$ | Locations | $\Gamma, m \vdash \eta : \sigma$ |
| Messages | $\Gamma \vdash M : \tau$ | Actions | $\Gamma, \rho_{deact}, \rho_{act}, m, u \vdash \pi : (\Gamma', \rho'_{act})$ |
| Processes | $\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}$ | | |

In Tables 3 and 4, let $\Gamma$ be a typing environment mapping message identifiers to basic types and port names to communication types. We further require that $\Gamma(m)$ be an ambient type for ambient names, and $\Gamma(i)$ be a capability type for capability variables (the two sorts of message identifiers). The typing judgment for a process is of the form $\Gamma, \rho_{deac}, m, u \vdash P : \rho_{act}$, where $\Gamma$ is the typing environment for free message identifiers and port names, $m$ is the assumed surrounding ambient, $u$ is the current user (owner of $m$), $\rho_{deact}$ is the set of roles that the process may at any time in its computation safely deactivate, and $\rho_{act}$ is the set of "currently active" roles. The judgments for the other syntactic categories are similar.

Typing of ambients and capabilities is mostly as would be expected. One should note that when typing these, one may use a more restrictive set of roles than is allowed by the typing environment. Thus, when typing a path, we may use the intersection of all the roles of the ambients in the path. However, we should note that the only communication type of the right-most component in a path is used for the communication type of the whole path. To type local communication, we use the ambient assumed as the surrounding ambient, and we want no restrictions on reading or writing. However, it is important that we maintain the restrictions on the types of data transmitted. We could violate the security policy if we omitted the type checks on messages locally communicated, because we potentially could send a capability with one security policy, but receive it with a different one.

The rules for typing actions and processes appear in Table 4. Actions are the basic unit of work in processes. They have the potential for changing the set of variables in scope, the current position and hence the current authorizing policy, and the set of activated roles. Thus the type of an action is a tuple of the revised typing environment, the revised authorizing policy, and the revised set of activated roles. Capabilities change the current location and introduce a new port, while co-capabilities only introduce a new port. Activation adds a new role, if it is allowed by the policy, and deactivation removes it provided it is in the set of roles safe for deactivation. Inputting a message introduces a tuple of new message variables. Outputting a message does not change the typing environment.

Processes are the outermost level of syntax. The main rules to note are those for prefixes and ambients (Prefixing and Ambient in Table 4). For prefixes, we must type the action at the head to derive a new typing environment, new authorizing policy, and a new set of active roles, and then use these instead of the originals to check the remaining process. The typing for an ambient throws away the surrounding ambient information and checks the ambient in isolation. Since an ambient may travel into other ambients with unknown active roles, an ambient must be secure relative to the context it carries with itself.

### *3.1 Typing Lemmas*

Before moving on to operational semantics, let us collect a few useful lemmas about the typing relation described in this section.

Capabilities:

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathsf{cap}(\rho_{in}, \sigma) \\ (\rho_{act} - \rho_{deact}) \cap \rho_{in} \neq \emptyset\end{array}}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, m, u \\ \vdash C(c{:}\sigma) : (\Gamma + (c{:}\sigma), \rho_{act})\end{array}}$$

Co-Capabilities:

$$\frac{\Gamma \vdash m : \mathsf{amb}(\rho_{in}, \sigma)}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, m, u \\ \vdash K(c{:}\sigma) : (\Gamma + (c{:}\sigma), \rho_{act})\end{array}}$$

Activation:

$$\frac{\mathsf{r} \in \mathit{UserPolicy}(u, \rho_{act})}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, m, u \\ \vdash \mathsf{activate}\langle \mathsf{r} \rangle : (\Gamma, \rho_{act} \cup \{\mathsf{r}\})\end{array}}$$

Deactivation:

$$\frac{\mathsf{r} \in \rho_{deact}}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, , m, u \\ \vdash \mathsf{deactivate}\langle \mathsf{r} \rangle : (\Gamma, \rho_{act} - \{\mathsf{r}\})\end{array}}$$

Input:

$$\frac{\begin{array}{c} m \notin \{x_1, \ldots, x_k\} \\ \Gamma, m \vdash \eta : (\rho_r, \rho_w, \tau) \\ (\rho_{act} - \rho_{deact}) \cap \rho_r \neq \emptyset \end{array}}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, m, u \\ \vdash (x_1, \ldots, x_k)^\eta : (\Gamma + \Sigma_{i=1}^{k}\{x_i{:}\tau\}, \rho_{act})\end{array}}$$

Output:

$$\frac{\begin{array}{c} \Gamma, m \vdash \eta : (\rho_r, \rho_w, \tau) \\ \Gamma \vdash M_i : \tau \quad i = 1, \ldots, k \\ (\rho_{act} - \rho_{deact}) \cap \rho_w \neq \emptyset \end{array}}{\begin{array}{l}\Gamma, \rho_{deact}, \rho_{act}, m, u \\ \vdash \langle M_1, \ldots, M_k \rangle^\eta : (\Gamma, \rho_{act})\end{array}}$$

Composition:

$$\frac{\begin{array}{c}\Gamma, \rho_{deact}, m, u \vdash P_1 : \rho_{act} \\ \Gamma, \rho_{deact}, m, u \vdash P_2 : \rho_{act}\end{array}}{\Gamma, \rho_{deact}, m, u \vdash P_1 \mid P_2 : \rho_{act}}$$

Replication:

$$\frac{\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}}{\Gamma, \rho_{deact}, m, u \vdash !P : \rho_{act}}$$

Nil:

$$\frac{}{\Gamma, \rho_{deact}, m, u \vdash \mathbf{0} : \rho_{act}}$$

Name Restriction:

$$\frac{nc \neq m \quad \Gamma + (nc{:}\varphi), \rho_{deact}, m, u \vdash P : \rho_{act}}{\Gamma, \rho_{deact}, m, u \vdash \boldsymbol{\nu}(nc{:}\varphi)P : \rho_{act}}$$

Prefixing:

$$\frac{\begin{array}{c}\Gamma, \rho_{deact}, \rho_{act}, m, u \vdash \pi : (\Gamma', \rho'_{act}) \\ \Gamma', \rho_{deact}, m, u \vdash P : \rho'_{act}\end{array}}{\Gamma, \rho_{deact}, m, u \vdash \pi.P : \rho_{act}}$$

Ambient:

$$\frac{m \in \mathbf{dom}(\Gamma) \qquad \Gamma, \rho'_{deact}, m, v \vdash P : \rho_m}{\Gamma, \rho_{deact}, m', u \vdash m_v[P]@\rho_m : \rho_{act}}$$

Table 4
Well-typed Actions and Processes

**Lemma 1 (Port Weakening)** *If* $\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}$ *and* $c \notin \boldsymbol{fp}(P)$ *then* $\Gamma + (c : \sigma), \rho_{deact}, m, u \vdash P : \rho_{act}$.

**PROOF.** *Outline.* It follows by induction on the derivation of $\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}$, using that $\Gamma + (c : \sigma) + (c : \sigma') = \Gamma + (c : \sigma')$ and $\Gamma + (c : \sigma) + (c' :$

$\sigma') = \Gamma + (c' : \sigma') + (c : \sigma)$ for $c \neq c'$. Similar results for the other kinds of judgements follow using the same argument.

A similar result holds for variable weakening.

**Lemma 2 (Port Substitution)** *If $\Gamma + (c : \sigma), \rho_{deact}, m, u \vdash P : \rho_{act}$ and $c' \notin (\mathit{fp}(P) - \{c\})$ then $\Gamma + (c' : \sigma), \rho_{deact}, m, u \vdash P\{c := c'\} : \rho_{act}$*

**Lemma 3 (Message Substitution)** *Let $x_1, \ldots, x_k$ be variables, and $M_1, \ldots, M_k$ be messages we wish to simulataneously substitute for $x_1, \ldots, x_k$. Let $\Gamma$ and $\Gamma'$ be two environments such that for all port names $c$, we have $\Gamma(c) = \Gamma'(c)$, and for all variables $y \notin \{x_1, \ldots, x_k\}$, we have $\Gamma(y) = \Gamma'(y)$. Also assume $\Gamma(x_i) = \tau_i$ and $\Gamma' \vdash M_i : \tau_i$ for all $i = 1, \ldots, k$. If $\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}$ then $\Gamma', \rho_{deact}, m\{x_i := M_i \,|i = 1, \ldots, k\}, u \vdash P\{x_i := M_i \,|i = 1, \ldots, k\} : \rho_{act}$.*

**Lemma 4 (Role Expansion)** *If $\mathsf{UserPolicy}$ is monotonically increasing in its role set argument (i.e., for all $\rho$, $\rho'$, $\rho \subseteq \rho'$ implies $\mathsf{UserPolicy}(u, \rho) \subseteq \mathsf{UserPolicy}(u, \rho')$), then $\Gamma, \rho_{deact}, m, u \vdash P : \rho'$, for all $\rho$, $\rho'$ with $\rho \subseteq \rho'$.*

## 4 Operational Semantics

Our goal in defining the static type system given in Section 3 is to enable us to prove that if a process type checks with a given set of roles, then it will never attempt an action that it is not authorized to perform when executed in a state where all the roles in the set have previously been activated. To this end, we define two transition semantics for our language, one with dynamic security checks and one without. For the untyped semantics, we have a form of subject reduction. We also have that, if a process type checks, then it reduces to another process in the untyped transition system if and only if it reduces to that process in the typed transition system.

### *4.1 Untyped Transition Semantics*

The untyped transition semantics is given in Table 5. It is worth noting that almost all the reduction rules explicitly mention a context containing an ambient, except for the rule for LOCAL communication.

The rules for ambient movement (ENTER and EXIT) are the most complicated. For an ambient to ENTER another the two ambients must be directly in parallel with each other, the first ambient must contain a process requesting entrance to the second, and the second ambient must have a process allowing the entrance. If these conditions are met, then the request and permission are

ENTER: $m_u[\mathsf{in}\,n\,(c{:}\sigma).P_1 \mid P_2]@\rho_m \mid n_v[\overline{\mathsf{in}}\,(c'{:}\sigma').P_3 \mid P_4]@\rho_n \Rightarrow$

$\boldsymbol{\nu}(c''{:}\sigma)n_v[m_u[P_1\{c := c''\} \mid P_2]@\rho_m \mid P_3\{c' := c''\} \mid P_4]@\rho_n$

EXIT: $p_w[n_v[m_u[\mathsf{out}\,p\,(c{:}\sigma).P_1 \mid P_2]@\rho_m \mid P_3]@\rho_n \mid \overline{\mathsf{out}}\,(c'{:}\sigma').P_4 \mid P_5]@\rho_p \Rightarrow$

$\boldsymbol{\nu}(c''{:}\sigma)p_w[m_u[P_1\{c := c''\} \mid P_2]@\rho_m \mid n_v[P_3]@\rho_n \mid P_4\{c := c''\} \mid P_5]@\rho_p$

where $c''$ is assumed to be a fresh variable in each transition above

ACTIVATE: $m_u[(\mathsf{activate}\langle \mathsf{r}\rangle.P) \mid R]@\rho \Rightarrow m_u[P \mid R]@(\rho \cup \{r\})$

DEACTIVATE: $m_u[(\mathsf{deactivate}\langle \mathsf{r}\rangle.P) \mid R]@\rho \Rightarrow m_u[P \mid R]@(\rho - \{r\})$

LOCAL: $\langle M_1,\dots,M_k\rangle^{\star}.\,P \mid (x_1,\dots,x_k)^{\star}.\,R \Rightarrow P \mid R\{x_i := M_i \,|\, i = 1\dots k\}$

TO CHILD (↓): $m_u[\langle M_1,\dots,M_k\rangle^{\downarrow c}.\,P_1 \mid n_v[(x_1,\dots,x_k)^{\uparrow c}.P_2 \mid R_1]@\rho_n \mid R_2]@\rho_m$

$\Rightarrow m_u[P_1 \mid n_v[P_2\{x_i := M_i|i = 1\dots k\} \mid R_1]@\rho_n \mid R_2]@\rho_m$

TO PARENT (↑): $n_v[m_u[\langle M_1,\dots,M_k\rangle^{\uparrow c}.P_1 \mid R_1]@\rho_m \mid (x_1,\dots,x_k)^{\downarrow c}.\,P_2 \mid R_2]@\rho_n$

$\Rightarrow n_v[m_u[P_1 \mid R_1]@\rho_m \mid P_2\{x_i := M_i|i = 1\dots k\} \mid R_2]@\rho_n$

COMPOSITION: $\dfrac{P_1 \Rightarrow R_1}{P_1 \mid P_2 \Rightarrow R_1 \mid P_2}$ RESTRICTION: $\dfrac{P \Rightarrow R}{\boldsymbol{\nu}(nc{:}\varphi)P \Rightarrow \boldsymbol{\nu}(nc{:}\varphi)R}$

AMBIENT: $\dfrac{P \Rightarrow R}{m_u[P]@\rho \Rightarrow m_u[Q]@\rho}$ STRUCT: $\dfrac{P' \equiv P \;\; P \Rightarrow R \;\; R \equiv R'}{P' \Rightarrow R'}$

Table 5
Simple Transition System

consumed and the resulting first ambient enters the resulting second ambient. Upon entrance, a fresh communication port is created for the two ambients to share (See rules CAPABILITIES and CO-CAPABILITIES in Table 4). The type of the port is determined by the ambient being entered. For an EXIT action, the conditions are the same except for the positioning of the ambients: the one requesting to exit must be inside an ambient which in turn is inside the ambient to which the first wishes to exit. The rules for activation and deactivation cause the addition or deletion of the given role from the role set of the surrounding ambient. The rules for communication cause the appropriate substitution when the communicating parties are appropriately positioned. It is worth noting that local communication is expressly not between ambients, but between ordinary processes, corresponding to communication in the $\pi$-calculus. In addition to the above rules for top-level reduction, there is a rule allowing us to descend through compositions, restrictions, and ambients to find a process capable of reducing. In particular, it is worth noting that an ambient within another ambient may keep computing, even while the outer ambient is blocked. To apply any of these rules, we may substitute for a process any process which is structurally equivalent to the original one.

## *4.2 Typed Transition Semantics*

In this section we briefly sketch the transition semantics with runtime type checks. For more details and proofs of the results in this section, please see [13]. The rules of the semantics augment the processes to be evaluated with a typing context in which the evaluation is to take place. This context comprises a typing environment, $\Gamma$, as in Section 3, and a basic type, $\tau$. As usual, the typing environment supplies us with the types for free ambient names and ports occurring in our process. The basic type is the type of a message that can be locally communicated at top level. We do not need read and write policies, because there are no security checks on local communication. The typed reduction relation transforms a process and its context into a new process in a new context. If we ignore the context, including the premises concerning it, then we get the untyped system in the previous section. The typing environment and basic type are the extra information we need to carry to do dynamic security checks.

The rules for the typed transition semantics can be grouped into four types: mobility, communication, role change, and structural. As the rules in each group are quite similar, we give one from each group here. For the remaining rules, we refer the reader to [13].

Since the reductions on the processes are the same as in the untyped transition semantics, we will focus on the security checks and the transformations to the typing environment and basic type. When one ambient enters into or exits to another, we need to know that the entering/exiting ambient has an appropriate role activated authorizing it to enter, and we need to establish a shared communication port sending and receiving messages of a type specified by the host ambient. The new communication port needs to be added to the typing environment. (See Exit in Table 6.) The side conditions for the rule for Enter are comparable to those for Exit.

The next theorem shows that the typed transition semantics is a refinement of the untyped transition semantics.

**Theorem 5** *Let $P$ and $P'$ be processes, $\Gamma$ and $\Gamma'$ be typing environments, and $\tau$ and $\tau'$ be basic types. If $(\Gamma, \tau) \rhd P \longrightarrow (\Gamma', \tau') \rhd P'$, then $\tau = \tau'$, $\Gamma = \Gamma'$, and $P \Rightarrow P'$.*

**PROOF.** The proof is by induction on the derivation of $(\Gamma, \tau) \rhd P \longrightarrow (\Gamma', \tau') \rhd P'$. A quick inspection shows the all the rules for typed transition have the same environment $\Gamma$ and same basic type $\tau$ on the left of the transition as on the right in their conclusion. To show that $P \Rightarrow P'$, we use that the only typed transition rules having a typed transition premise are the struc-

Exit:

$$\frac{\begin{array}{c}\Gamma(p) = \mathsf{amb}(\rho_{in}, \sigma) \qquad \rho_m \cap \rho_{in} \neq \emptyset \\ c'' \notin ((\mathtt{fp}(P_1) - \{c\}) \cup \mathtt{fp}(P_2) \cup \mathtt{fp}(P_3) \cup (\mathtt{fp}(P_4) - \{c'\}) \cup \mathtt{fp}(P_5))\end{array}}{\begin{array}{c}(\Gamma, \tau) \rhd p_w[n_v[m_u[\mathsf{out}\, p\, (c{:}\sigma).P_1 \mid P_2]@\rho_m \mid P_3]@\rho_n \mid \overline{\mathsf{out}}\,(c'{:}\sigma).P_4 \mid P_5]@\rho_p \longrightarrow \\ (\Gamma, \tau) \rhd \boldsymbol{\nu}(c''{:}\sigma)p_w[m_u[P_1\{c := c''\} \mid P_2]@\rho_m \mid n_v[P_3]@\rho_n \mid P_4\{c := c''\} \mid P_5]@\rho_p\end{array}}$$

To Parent (↑):

$$\frac{\Gamma(c) = (\rho_r, \rho_w, \tau') \quad \rho_m \cap \rho_w \neq \emptyset \quad \rho_n \cap \rho_r \neq \emptyset \quad \Gamma \vdash M_i : \tau' \quad i = 1, \ldots, k}{\begin{array}{c}(\Gamma, \tau) \rhd n_v[m_u[\langle M_1, \ldots, M_k\rangle^{\uparrow c}.P_1 \mid R_1]@\rho_m \mid (x_1, \ldots, x_k)^{\downarrow c}.\, P_2 \mid R_2]@\rho_n \longrightarrow \\ (\Gamma, \tau) \rhd n_v[m_u[P_1 \mid R_1]@\rho_m \mid P_2\{x_i := M_i | i = 1 \ldots k\} \mid R_2]@\rho_n\end{array}}$$

Activate:

$$\frac{r \in \mathit{UserPolicy}(u, \rho)}{(\Gamma, \tau) \rhd m_u[(\mathsf{activate}\langle \mathsf{r}\rangle P) \mid R]@\rho \longrightarrow (\Gamma, \tau) \rhd m_u[P \mid R]@(\rho \cup \{r\})}$$

Restriction :

$$\frac{(\Gamma + \{nc : \varphi\}, \tau) \rhd P \longrightarrow (\Gamma' + \{nc : \varphi\}, \tau) \rhd R}{(\Gamma, \tau) \rhd \boldsymbol{\nu}(nc{:}\varphi)P \longrightarrow (\Gamma', \tau) \rhd \boldsymbol{\nu}(nc{:}\varphi)R}$$

Table 6
Typed Transitions

tural rules. For the non-structural rules, erasing the environments and basic types leaves us with the corresponding rules for the untyped transition system. For the structural rules, erasing the environments and types from both the premises and conclusions yields the corresponding rules from the untyped system.

Theorem 6 can be seen as the combination of two different results. The first one is the traditional Subject Reduction result that states that evaluation preserves typing, while the second result tells us that runtime type checks can be omitted on well typed processes. A side-effect of the latter is that if a process type checks, there is no runtime significance to activation and deactivation, and these operations could be removed after type-checking as an optimization.

**Theorem 6 (Subject Reduction and Runtime Typechecks)** *Let $P$ and $Q$ be processes, $m$ be an ambient name, $u$ be a user, $\rho_{deact}$ and $\rho_{act}$ be sets of roles, and let $\Gamma$ be a typing environment such that $m \in$ **dom**$(\Gamma)$. Let $\Gamma(m) = \mathsf{amb}(\rho_{in}, (\rho_r, \rho_w, \tau))$, If $\Gamma$, $\rho_{deact}$, $m$, $u \vdash P : \rho_{act}$ and $P \Rightarrow Q$, then $(\Gamma, \tau) \rhd P \longrightarrow (\Gamma, \tau) \rhd Q$, and if in addition* UserPolicy *is monotonically increasing in role sets, then $\Gamma$, $\rho_{deact}$, $m$, $u \vdash Q : \rho_{act}$.*

**PROOF.** We proceed by induction on the derivation of $P \Rightarrow Q$ and by case analysis on the last rule used. The cases fall into four main categories: mobility, communication, role change, and structural. The proofs are fairly similar in each category, so we will describe one case from each category.

Exit: There exists a fresh variable $c''$ (not occurring free in $P$ and $Q$) such that

$$P = p_w[n_v[m'_{u'}[\mathsf{out}\, p\,(c\!:\!\sigma).P_1 \mid P_2]@\rho_{m'} \mid P_3]@\rho_n \mid \overline{\mathsf{out}}\,(c'\!:\!\sigma).P_4 \mid P_5]@\rho_p$$

$$\Rightarrow \boldsymbol{\nu}(c''\!:\!\sigma)p_w[n_v[P_3]@\rho_n \mid m'_{u'}[P_1\{c := c''\}]@\rho_{m'} \mid P_4\{c' := c''\} \mid P_5]@\rho_p = Q$$

and $\Gamma,\ \rho_{deact},\ m,\ u \vdash P : \rho_{act}$.

By inversion of the typing rules Ambient and Composition we have $m'$, $n$, and $p \in \textbf{dom}(\Gamma)$ and there exist role sets $\rho'_{deact}$, $\rho''_{deact}$, and $\rho'''_{deact}$ such that

$$\Gamma,\ \rho'_{deact},\ m',\ u' \vdash \mathsf{out}\, p\,(c\!:\!\sigma).P_1 : \rho_{m'} \qquad \Gamma,\ \rho'_{deact},\ m',\ u' \vdash P_2 : \rho_{m'}$$

$$\Gamma,\ \rho''_{deact},\ n,\ v \vdash P_3 : \rho_n$$

$$\Gamma,\ \rho'''_{deact},\ p,\ w \vdash \overline{\mathsf{out}}\,(c'\!:\!\sigma').P_4 : \rho_p \qquad \Gamma,\ \rho'''_{deact},\ p,\ w \vdash P_5 : \rho_p$$

By inversion of the rules Prefixing, Capabilities, Co-Capabilities, Exit To, and Ambient Name we have that $\sigma = \sigma'$ and there exist role sets $\rho'_{in}$, $\rho''_{in}$, $\rho'''_{in}$ such that $\Gamma(m') = \mathsf{amb}(\rho'_{in}, \sigma'')$, $\Gamma(n) = \mathsf{amb}(\rho''_{in}, \sigma''')$, $\Gamma(p) = \mathsf{amb}(\rho'''_{in}, \sigma)$, $\Gamma + (c\ :\ \sigma),\ \rho'_{deact},\ m',\ u' \vdash P_1 : \rho_{m'}$, $\Gamma + (c'\ :\ \sigma),\ \rho'''_{deact},\ p,\ w \vdash P_4 : \rho_p$, and$(\rho_{m'} - \rho'_{deact}) \cap \rho'''_{in} \neq \emptyset$, and hence $\rho_{m'} \cap \rho'''_{in} \neq \emptyset$. Thus, since $c''$ was chosen fresh for $P$, by the Exit rule from the typed transition system, we have

$$(\Gamma, \tau) \triangleright p_w[n_v[m'_{u'}[\mathsf{out}\, p\,(c\!:\!\sigma).P_1 \mid P_2]@\rho_{m'} \mid P_3]@\rho_n \mid \overline{\mathsf{out}}\,(c'\!:\!\sigma).P_4 \mid P_5]@\rho_p \longrightarrow$$

$$(\Gamma, \tau) \triangleright \boldsymbol{\nu}(c''\!:\!\sigma)p_w[n_v[P_3]@\rho_n \mid m'_{u'}[P_1\{c := c''\}]@\rho_{m'} \mid P_4\{c' := c''\} \mid P_5]@\rho_p$$

and thus that $(\Gamma, \tau) \triangleright P \longrightarrow (\Gamma, \tau) \triangleright Q$.

Since we have $\Gamma + (c : \sigma),\ \rho'_{deact},\ m',\ u' \vdash P_1 : \rho_{m'}$ and $\Gamma + (c' : \sigma),\ \rho'''_{deact},\ p,\ w \vdash P_4 : \rho_p$ by Lemma 2, since $c''$ is fresh for $P$ we have

$$\Gamma + (c'' : \sigma),\ \rho'_{deact},\ m',\ u \vdash P_1\{c := c''\} : \rho_{m'},$$

and

$$\Gamma + (c'' : \sigma),\ \rho'''_{deact},\ p,\ w \vdash P_4\{c' := c''\} : \rho_p.$$

Also because $c''$ is fresh for $P$, by Lemma 1, we have

$$\Gamma + (c'' : \sigma),\ \rho'_{deact},\ m',\ u \vdash P_2 : \rho_{m'}, \quad \Gamma + (c'' : \sigma),\ \rho''_{deact},\ n,\ v \vdash P_3 : \rho_{m'},$$

$$\Gamma + (c'' : \sigma),\ \rho'''_{deact},\ p,\ w \vdash P_5 : \rho_p.$$

By repeated application of COMPOSITION and AMBIENT, and finally an application of NAME RESTRICTION we have $\Gamma$, $\rho_{deact}$, $m$, $u \vdash Q : \rho_{act}$. This finishes the case for EXIT. The other mobility case, ENTER, follows a similar argument, but with fewer uses of AMBIENT and COMPOSITION.

TO PARENT: In this case we have

$$P = n_v[m'_{u'}[\langle M_1, \ldots, M_k \rangle^{\uparrow c}.P_1 \mid P_2]@\rho_{m'} \mid (x_1, \ldots, x_k)^{\downarrow c}.\,P_3 \mid P_4]@\rho_n,$$

$$Q = n_v[m'_{u'}[P_1 \mid P_2]@\rho_{m'} \mid P_3\{x_i := M_i | i = 1 \ldots k\} \mid P_4]@\rho_n.$$

$$\Gamma, \rho_{deact}, m, u \vdash P : \rho_{act}.$$

By inversion of the typing rules AMBIENT, COMPOSITION, PREFIXING, OUTPUT, INPUT, PARENT PORT, and CHILD PORT, we have $m'$ and $n \in \mathbf{dom}(\Gamma)$ and there exist role sets $\rho'_{deact}$, $\rho''_{deact}$, $\rho'_r$, and $\rho'_w$, and a basic type $\tau'$ such that

$$\begin{array}{ll}
\Gamma, \rho'_{deact}, m', u' \vdash P_1 : \rho_{m'}, & \Gamma + \Sigma_{i=1}^{k}\{x_i : \tau'\}, \rho''_{deact}, n, v \vdash P_3 : \rho_n \\
\Gamma, \rho'_{deact}, m', u' \vdash P_2 : \rho_{m'} & \Gamma, \rho''_{deact}, n, v \vdash P_4 : \rho_n \\
\Gamma \vdash M_i : \tau' \ i = 1, \ldots, k & n \notin \{x_1, \ldots, x_k\} \\
\Gamma(c) = (\rho'_r, \rho'_w, \tau') & (\rho_n - \rho''_{deact}) \cap \rho'_r \neq \emptyset \\
(\rho_{m'} - \rho'_{deact}) \cap \rho'_w \neq \emptyset &
\end{array}$$

Therefore, we have $\rho_{m'} \cap \rho'_w \neq \emptyset$ and $\rho_n \cap \rho'_r \neq \emptyset$, and thus by the typed transition rule TO PARENT we have

$$(\Gamma, \tau) \rhd P \longrightarrow (\Gamma, \tau) \rhd Q$$

which is the first half of what was to be shown in this case.

Since $\Gamma$ and $\Gamma + \Sigma_{i=1}^{k}\{x_i : \tau'\}$ only differ on $\{x_1, \ldots, x_k\}$, and $\Gamma \vdash M_i : \tau'$ for all $i = 1 \ldots k$, and since $n \notin \{x_1, \ldots, x_k\}$, by Lemma 3, we have that

$$\Gamma, \rho''_{deact}, n, v \vdash P_3\{x_i := M_i\, | i = 1 \ldots k\} : \rho_n.$$

By application of COMPOSITION and AMBIENT we have

$$\Gamma, \rho''_{deact}, n, v \vdash m'_{u'}[P_1 \mid P_2]@\rho_{m'} : \rho_n.$$

By further applications of COMPOSITION and a final application of AMBIENT, we have $\Gamma, \rho_{deact}, m, u \vdash Q : \rho_{act}$, which is the second half of what we needed to show in this case.

This finishes the case TO PARENT. The case TO CHILD is almost identical. The last communication case, LOCAL is similar, but somewhat simpler since it does not need to use the AMBIENT rule. Also in the LOCAL case, channel types are learned through a combination of the LOCAL and AMBIENT NAME typing rules, instead of PARENT/CHILD PORT rules.

ACTIVATE: We have

$$P = m'_{u'}[(\mathsf{activate}\langle \mathsf{r}\rangle.P_1) \mid P_2]@\rho_{m'} \Rightarrow m'_u[P_1 \mid P_2]@(\rho_{m'} \cup \{r\}) = Q$$

and

$$\Gamma, \rho_{deact}, m, u \vdash m'_{u'}[(\mathsf{activate}\langle \mathsf{r}\rangle.P_1) \mid P_2]@\rho_{m'} : \rho_{act}$$

By inversion of the typing rules AMBIENT, COMPOSITION, PREFIXING and ACTIVATION $m' \in \mathbf{dom}(\Gamma)$ and there exists a role set $\rho'_{deact}$ such that $\Gamma, \rho'_{deact}, m', u' \vdash P_2 : \rho_{m'}$ and $\Gamma, \rho'_{deact}, m', u' \vdash P_1 : \rho_{m'} \cup \{r\}$ and $r \in \mathit{UserPolicy}(u', \rho_{m'})$. Hence, by the typed transition rule ACTIVATE, we have

$$(\Gamma, \tau) \rhd m'_{u'}[(\mathsf{activate}\langle \mathsf{r}\rangle.P_1) \mid P_2]@\rho_{m'} \longrightarrow (\Gamma, \tau) \rhd m'_{u'}[P_1 \mid P_2]@(\rho_{m'} \cup \{r\}).$$

It remains to show $\Gamma$, $\rho_{deact}$, $m$, $u \vdash m'_{u'}[P_1 \mid P_2]@(\rho_{m'} \cup \{r\}) : \rho_{act}$. Since *UserPolicy* is monotonically increasing in role sets, by Lemma 4, we have $\Gamma, \rho'_{deact}, m', u' \vdash P_2 : \rho_{m'} \cup \{r\}$. Therefore, applying the typing rules COMPOSITION and AMBIENT, we have $\Gamma, \rho_{deact}, m, u \vdash m'_{u'}[P_1 \mid P_2]@(\rho_{m'} \cup \{r\}) : \rho_{act}$, as was to be shown.

The case for DEACTIVATE is similar, but requires observing that if a process type-checks with a given role set, then we can remove any element of the role set $\rho_{deact}$ from the given role set, and the process will still type-check with respect to it.

RESTRICTION: In this case, $P = \boldsymbol{\nu}(nc{:}\varphi)P_1$ and $Q = \boldsymbol{\nu}(nc{:}\varphi)Q_1$ and $P_1 \Rightarrow Q_1$. By inversion of the typing rule NAME RESTRICTION, we have that $nc \neq m$ and $\Gamma + (nc : \varphi), \rho_{deact}, m, u \vdash P_1 : \rho_{act}$. By the induction hypothesis, we have that $(\Gamma + (nc : \varphi), \tau) \rhd P_1 \longrightarrow (\Gamma + (nc : \varphi), \tau) \rhd Q_1$, and $\Gamma + (nc : \varphi), \rho_{deact}, m, u \vdash Q_1 : \rho_{act}$. Therefore by the typed transition rule RESTRICTION we have $(\Gamma, \tau) \rhd \boldsymbol{\nu}(nc{:}\varphi)P_1 \longrightarrow (\Gamma, \tau) \rhd \boldsymbol{\nu}(nc{:}\varphi)Q_1$ and by the typing rule NAME RESTRICTION we have $\Gamma, \rho_{deact}, m, u \vdash \boldsymbol{\nu}(nc{:}\varphi)Q_1 : \rho_{act}$, as was to be shown.

In the cases for the other structural rules, COMPOSITION, RESTRICTION, AMBIENT, and STRUCT EQUIV, the proof proceeds in a similar manner using the induction hypothesis.

The typed transition semantics developed in this section was primarily introduced as a vehicle to formalize the benefit of static type checking. It is worth noting that this semantics is of value in its own right. The static rules are

| Ambient | User | Roles | Ambient | User | Roles |
|---|---|---|---|---|---|
| laptop | ProfSue | {instructor} | mail | ProfSue | {faculty_mail} |
| | Dan | {student} | | Dan | {student_mail} |
| | Chuck | {} | | Chuck | {} |
| classroom / lounge / Univ | ProfSue | {} | answer | ProfSue | {instructor} |
| | Dan | {} | | Dan | {student} |
| | Chuck | {sys_admin} | | Chuck | {} |

$$\Gamma(\mathsf{classroom}) = \mathsf{amb}(\{\mathsf{student}, \mathsf{instructor}, \mathsf{faculty\_mail}\}, \_)$$
$$\Gamma(\mathsf{Univ}) = \Gamma(\mathsf{lounge}) = \mathsf{amb}(\{\mathsf{student}, \mathsf{student\_mail}, \mathsf{instructor}, \mathsf{faculty\_mail}\}, \_)$$

Table 7
Wizbrau Security Policy

predicated on static access to the information as to which roles are granted access to which resources. With the typed transition semantics, we can still perform security checks even in a situation where the control policy is only known at runtime.

## 5 Example: The University of Wizbrau

To see the utility of the calculus discussed in this paper, let us see how we could use it to express an example where we need to combine mobility with localized checking of access authorization to local resources. Recall the example outlined in the introduction of the university with classrooms where students' access to the Internet is more limited than that of the instructor. Let $\{\mathsf{ProfSue}, \mathsf{Dan}, \mathsf{Chuck}\}$ be users representing the instructor in the class, a student in the class, and the systems administrator for the classrooms in the university. There are five roles available, student, student_mail, instructor, faculty_mail and sys_admin. To send mail, one should use an ambient named mail. To use the automated class response system, one should use an ambient named answer. The laptops of the students and the instructor will be represented by ambients named laptop, the classroom ambient will be named classroom and the student lounge ambient will be named lounge. The classroom and the student lounge are part of the Univ ambient.

The user policy and a partial initial typing environment are given in Table 7. For this example, the user policy does not depend on the currently active policies, and they are omitted in the table. For brevity, we will also omit communication types from the typing environment description.

The basic program for both the classroom and the student lounge is the same. It allows other ambients to come and go, provided they have the right roles activated. For those entering, no communication is done. For those exiting, a path to a router is provided. The program for the instructor is to enter the classroom and send some mail, in parallel with some other work. The student enters the classroom, answers some questions, but also wants to send some mail. Here we can examine some options that fail, together with one that works. The formal processes are listed below. To assist in examining the possibilities for Dan, his process will be parameterized by the role activated for the mail ambient and the ambient to which the mail ambient tries to exit.

$$\mathsf{ClassRoom} = \mathsf{classroom}_{\mathsf{Chuck}}[!\overline{\mathsf{in}}\,(c).\mathbf{0} \mid !\overline{\mathsf{out}}\,(c').\langle \textit{path to router}\rangle.\mathbf{0}]@\{\}$$

$$\mathsf{Lounge} = \mathsf{lounge}_{\mathsf{Chuck}}[!\overline{\mathsf{in}}\,(c).\mathbf{0} \mid !\overline{\mathsf{out}}\,(c').\langle \textit{path to router}\rangle.\mathbf{0}]@\{\}$$

$$\begin{array}{l}\mathsf{mail\_amb}(user,\ role,\ amb)\ = \\ \quad \mathsf{mail}_{user}[\mathsf{activate}\langle role\rangle.\mathsf{out}\ amb\, c.(i)^{\uparrow c}.i(c').P_{DM}\,]@\{\}\end{array}$$

$$\begin{array}{l}\mathsf{ProfSuesLaptop} = \\ \quad \mathsf{laptop}_{\mathsf{ProfSue}} \left[\begin{array}{l}\mathsf{activate}\langle\mathsf{instructor}\rangle.\mathsf{in\ classroom}\, c. \\ (\mathsf{mail\_amb}(\mathsf{ProfSue},\ \mathsf{faculty\_mail},\ \mathsf{classroom}) \mid P)\end{array}\right] @\{\}\end{array}$$

$$\mathsf{answer\_amb} = \mathsf{answer}_{\mathsf{Dan}}[\mathsf{activate}\langle\mathsf{student}\rangle.\mathsf{out\ classroom}\, c.(i)^{\uparrow c}.i(c').P_{DA}\,]@\{\})$$

$$\begin{array}{l}\mathsf{DansLaptop}(role,\ amb) = \\ \quad \mathsf{laptop}_{\mathsf{Dan}} \left[\begin{array}{l}\mathsf{activate}\langle\mathsf{student}\rangle.\mathsf{in}\ classroom\, c.(!\mathsf{answer\_amb} \mid \\ \mathsf{mail\_amb}(\mathsf{Dan},\ role,\ amb) \mid \mathsf{out\ Univ}\, c.\mathsf{in\ lounge}\, c'.Q)\end{array}\right] @\{\}\end{array}$$

The total process, parameterized by the arguments for the ambient DansLaptop then is:

$$\mathsf{Univ}_{\mathsf{Chuck}}[\mathsf{ClassRoom} \mid \mathsf{Lounge} \mid \mathsf{ProfSuesLaptop} \mid \mathsf{DansLaptop}(role,\ amb)\,]@\{\}.$$

Using the untyped transition system, both ProfSuesLaptop and DansLaptop will enter the ClassRoom and will be able to send mail. More precisely, the untyped transition rules will allow the ambients $\mathsf{laptop}_{\mathsf{ProfSue}}$ and $\mathsf{laptop}_{\mathsf{Dan}}$ to enter the $\mathsf{classroom}_{\mathsf{Chuck}}$ ambient and send out their $\mathsf{mail}_{\mathsf{Dan}}$ and $\mathsf{mail}_{\mathsf{ProfSue}}$ ambients.

Using the typed transition system, both ProfSuesLaptop and DansLaptop still can enter the ClassRoom. However, when the $\mathsf{mail}_{\mathsf{Dan}}$ ambient attempts to exit DansLaptop to the classroom ambient, it will fail because $\mathsf{mail}_{\mathsf{Dan}}$ will not have an authorizing role activated. This is caught both statically by type checking and dynamically by run-time type checks in the typed transition system for the process where *role* is student_mail, and *amb* is classroom.

To understand how type checking and typed transitions proceed, we need to consider the restriction imposed by the user policy and the typing environment. The *UserPolicy* in Table 7 says, among other policy statements, that $\mathsf{mail}_{\mathsf{Dan}}$ can only activate student_mail, and the typing environment $\Gamma$ states that in order to enter classroom an ambient has to have at least one of the roles in {student, instructor, faculty_mail} activated. Therefore, the ambient $\mathsf{mail}_{\mathsf{Dan}}$ cannot enter the classroom ambient. By the same argument, the *UserPolicy* says that $\mathsf{mail}_{\mathsf{ProfSue}}$ can activate the role faculty_mail and enter the classroom ambient. The total process will type check if and only if $role =$ student_mail and $amb =$ lounge or Univ. If $role$ takes any other value, then the activation in mail will fail to type check because of the user policy. If $role =$ student_mail, then $amb$ cannot be classroom, since, by the typing environment, student_mail is not sufficient to grant entrance to classroom.

## 6 Related Work

For a variety of calculi for mobile and distributed systems that have emerged in recent years, access control was one of the primary concerns. The proposed access control mechanisms range from simple ones that use co-actions [22,28,3] allowing or denying all access to a particular location (and the resources it contains) to more refined ones that use different approaches: credentials to authorize the access [7], restricted groups [9,14], Mandatory Access Control mechanisms to constraint unauthorized access [5], and even "membranes" that specify security policies for controlling the access to a particular location [18].

The work most closely related to our study of RBAC for an ambient calculus is [4]. The authors define a distributed $\pi$-calculus (D-$\pi$) based on [19] with primitives to activate and deactivate roles. However, there is no notion of an individual privilege being disabled or enabled depending on the current location, and the domain topology is static: domains cannot move. In [20] Hennessy and Riely introduce a type system for a distributed version of the $\pi$-calculus for restricting the access of processes to resources based on the current location of the process. In this work, again the domain topology is static, and there is no direct connection to RBAC.

At the Symposium on Trustworthy Global Computing 2005 (TGC 2005), during his invited address, Matthew Hennessy presented a calculus for RBAC based on D-$\pi$. Unlike our system, his calculus has dependent types to avoid dynamic typechecks of the security policy.

The work of RBAC in [24,25] does not deal with the implementation of an RBAC mechanism in a given calculus as is the case in [4]. Instead they define a calculus to describe an RBAC security policy and how to answer queries to the security policy.

Various groups have developed methods for guaranteeing that specifications of RBAC systems are consistent. In [27], Schaad and Moffett discuss the application of formal methods for the development of specifications of a conflict-free role-based system. In [1] a formal language for the specification of role-based authorization constraints, including prohibition, is introduced. Bertino et al. [2] develop a logical framework for reasoning about access control models in general, including RBAC models.

## 7 Conclusions and Future Work

We defined $\mathbf{BACI_R}$, a boxed ambients calculus with Distributed Role-Based Access Control, where the privileges associated to processes change during computation and are determined by their location, their owners, the roles they have activated, and the security policy. The distributed nature of the RBAC mechanism comes from the fact that each ambient controls the security policy authorizing the entrance of ambients and each port specifies the security policy controlling the reading and writing privileges.

Our type system prevents two forms of security violations, those consisting of attempting to enter an ambient without proper authorization, and those consisting of trying to read or write from ports without the corresponding permissions. These security violations are controlled using roles, that can be dynamically activated and deactivated. The type system prevents security violating actions by those processes not vested with the required authorizing roles.

Our main contribution is the design of the first ambient calculus with a distributed RBAC mechanism where the location of a process conditions its mobility and its ability to communicate with other processes. Our main result in Theorem 6 shows that a well-typed program never violates the dynamic security checks.

Although the classroom example in the introduction and Section 5 is focused on Internet networking for a sense of location and communication, our Distributed RBAC mechanism should be applicable to other settings such as those arising from mobile telecommunications.

The area remains full of open and challenging problems. An interesting aspect to consider is the notion of trust in such a way that the access control policy governing the users' requests will further depend on whether the user is in a trusted or untrusted domain. Furthermore, RBAC can be enriched by placing order structures on roles (role hierarchies), constraints on roles such as mutual exclusion (no user may activate two given roles at the same time), combination of roles (two given roles have to be activated at the same time), and composition of roles (users having a given role are given another role).

Defining type systems to address these richer notions of RBAC is the subject of our ongoing and future research.

## 8 Acknowledgments

We are grateful to Pablo Garralda, Healfdene Goguen, and Mariangiola Dezani for illuminating discussions and comments on earlier drafts. We also appreciate Kaijun Tan for introducing us to the idea of combining RBAC with calculi for concurrency.

## References

[1] G. J. Ahn and R. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, 2000.

[2] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. In *Proc. of 6th SACMAT*, pages 41–52. ACM Press, 2001.

[3] Eduardo Bonelli, Adriana Compagnoni, Mariangiola Dezani-Ciancaglini, and Pablo Garralda. Boxed Ambients with Communication Interfaces (BACI). In *Proceedings Of The 29th International Symposium On Mathematical Foundations Of Computer Science (MFCS 2004) Prague, Czech Republic, Europe. 22-27 August 2004*, volume 3153 of *Lecture Notes In Computer Science*, pages 119–148, August 2004.

[4] C. Braghin, D. Gorla, and V. Sassone. Rôle-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155, 2006.

[5] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Reasoning about security in mobile ambients. In *CONCUR '01: Proceedings of the 12th International Conference on Concurrency Theory*, pages 102–120, London, UK, 2001. Springer-Verlag.

[6] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Access Control for Mobile Agents: The Calculus of Boxed Ambients. *ACM Transactions on Programming Languages and Systems*, 26(1):57–124, 2004.

[7] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication interference in mobile boxed ambients. In *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science, FST&TCS 2002*, volume 2556 of *LNCS*, pages 71–84. Springer, 2002.

[8] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication and Mobility Control in Boxed Ambients. *Information and Computation*, 202(1):39–86, August 2005.

[9] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient Groups and Mobility Types. In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *TCS'00*, volume 1872 of *Lecture Notes in Computer Science*, pages 333–347, Berlin, 2000. Springer-Verlag.

[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] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. Special Issue on Coordination, Daniel Le Métayer Editor.

[12] Adriana Compagnoni and Elsa Gunter. Types for security in a mobile world. In Rocco De Nicola and Davide Sangiorgi, editors, *Trustworthy Global Computing, International Symposium, TGC 2005, Edinburgh, UK, April 7-9, 2005*, volume 3705 of *Lecture Notes in Computer Science*, pages 75–97. Springer, 2005.

[13] Adriana Compagnoni, Elsa Gunter, and Philippe Bidinger. A role-based access control type system for boxed ambients. Technical Report UIUCDCS-R-2006-2753, University of Illinois at Urban-Champaign, 2006.

[14] Mario Coppo, Mariangiola Dezani-Ciancaglini, Elio Giovannetti, and Ivano Salvo. M3: Mobility Types for Mobile Processes in Mobile Ambients. In James Harland, editor, *CATS'03*, volume 78 of *ENTCS*. Elsevier, 2003.

[15] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[16] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[17] Pablo Garralda and Adriana Compagnoni. Splitting Mobility and Communication in Boxed Ambients. In Maribel Fernandez and Ian Mackie, editors, *International Workshop on Developements in Computational Models (DCM 2005)*, ENTCS. Elsevier, 2005.

[18] D. Gorla, M. Hennessy, , and V. Sassone. Security policies as membranes in systems for global computing. In *Foundations of Global Ubiquitous Computing, FGUC 2004*, ENTCS, 2004.

[19] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed system (extended abstract). In Andrew D. Gordon, editor, *FOSSACS'03*, volume 2620 of *LNCS*, pages 282–299, Berlin, 2003. Springer-Verlag.

[20] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

[21] Francesca Levi and Davide Sangiorgi. Controlling Interference in Ambients. *Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[22] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[23] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[24] Ninghui Li, William H. Winsborough, and John C. Mitchell. Distributed credential chain discovery in trust management: extended abstract. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 156–165. ACM Press, 2001.

[25] Ninghui Li, William H. Winsborough, and John C. Mitchell. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy*, page 123. IEEE Computer Society, 2003.

[26] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[27] Andreas Schaad and Jonathan D. Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 13–22. ACM Press, 2002.

[28] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. In Henri E. Bal, Boumediene Belkhouche, and Luca Cardelli, editors, *Internet Programming Languages*, volume 1686 of *Lecture Notes in Computer Science*, pages 47–77, Berlin, 1999. Springer-Verlag.