In Proc. of *17th Computer Security Foundations Workshop (CSFW'04)*, pagg. 48–60. © IEEE.

A Distributed Calculus for Rôle-Based Access Control

Chiara Braghin
Dip. di Informatica
Univ. Ca' Foscari di Venezia

Daniele Gorla
Dip. Sistemi e Informatica, Univ. di Firenze
Dip. Informatica, Univ. di Roma 'La Sapienza'

Vladimiro Sassone
University of Sussex

Abstract

Rôle-based access control (RBAC) is increasingly attracting attention because it reduces the complexity and cost of security administration by interposing the notion of rôle in the assignment of permissions to users. In this paper, we present a formal framework relying on an extension of the π calculus to study the behavior of concurrent systems in a RBAC scenario. We define a type system ensuring that the specified policy is respected during computations, and a bisimulation to equate systems. The theory is then applied to three meaningful examples, namely finding the 'minimal' policy to run a given system, refining a system to be run under a given policy (whenever possible), and minimizing the number of users in a given system without changing the overall behavior.

Introduction

Rôle-based access control (RBAC) [6, 17] has recently emerged as a widely accepted alternative to classical discretionary and mandatory access controls: a standard is currently under development by the National Institute of Standards and Technology (NIST) [7] and several commercial applications directly support some forms of RBAC, e.g., Oracle, Informix and Sybase in the field of commercial database management systems.

RBAC is a flexible and policy-neutral access control technology: it regulates the access of users to information and system resources on the basis of activities they need to execute in the system. The essence of RBAC lies with the notions of *user*, *rôle* and *permission*: users are authorized to use the permissions assigned to the rôles they belong to. More specifically, RBAC allows for a preliminary assignment of permissions to rôles (thus abstracting from which users will play the various rôles at run-time). A user may then establish multiple sessions, e.g., by signing on to the system, during which he activates a subset of rôles that he is a member of. This greatly simplifies system management, as it reduces the cost of administering access control policies, as well as making the administration process less error-prone. Anyway, the complexity of the models (e.g., in large systems the number of rôles can exceed hundreds or thousands) demands a structured approach to the analysis and design of such systems.

This paper aims at developing a foundational theory for system behaviors in a RBAC scenario; to the best of our knowledge this is the first attempt in this direction. Our reference model is the so-called RBAC96 model, introduced by Sandhu et al. in the seminal paper [17]. More advanced RBAC models include rôle hierarchies and constraints such as rôle mutual exclusion, separation of duty, delegation of authority and negative permissions. Our starting point is the π calculus [18], which provides very well-established mathematical tools for expressing concurrent and possibly distributed systems. Essentially, our idea is to equip the π calculus with a notion of users (i.e., named processes), with two new constructs for activation/deactivation of rôles, and with a way to grant permissions to rôles. This is accounted for by associating each process with a name representing a *user* and with a set ρ recording the rôles activated by the user during the current session. Hence, the term $r\{\!|\, P \,|\!\}_\rho$ represents a session of the user named r, running a process P with active rôles ρ. The calculus is completed by two constructs to model rôle's activation/deactivation, defined by the following reductions:

$$r\{\!|\, \mathbf{role}\, R.P \,|\!\}_\rho \longmapsto r\{\!|\, P \,|\!\}_{\rho\cup\{R\}}$$
$$r\{\!|\, \mathbf{yield}\, R.P \,|\!\}_\rho \longmapsto r\{\!|\, P \,|\!\}_{\rho-\{R\}}$$

Intuitively, when a user r activates a rôle R during a session, R must be added to the set of activated rôles ρ, and the remaining of the session P will be executed with the set ρ updated. Vice versa for the deactivation of R.

As an example, the following system

$$\begin{array}{l} \mathit{client}\{\!|\, \mathbf{role}\, \texttt{auth\_client}.\mathit{port\_80}\langle \mathit{index.html}\rangle.P \,|\!\}_\rho \\ \quad \| \ \mathit{server}\{\!|\, \mathit{port\_80}(x).Q \,|\!\}_{\rho'} \end{array}$$

models the interaction between a client and a HTTP server. The system contains two users, *client* and *server*, running in parallel. It may evolve as follows. First, user *client* activates the rôle `auth_client` by exercising the **role** action, which in practice would involve to authenticate herself by means of a secure certificate. Then, she sends the request to the HTTP server to the usual port 80, i.e., performs an output action on the channel *port_80*.

The introduction of named users immediately suggests the idea of a distributed system. In such systems, as e.g. the Internet, the notion of global, non-located channels as *port_80* is quite an abstraction over what is realistically achievable. We therefore use a notion of localized channels *à la* Dπ [10], where each channel is associated to a single user. Syntactically, we realize this feature by tagging output actions to specify the user (or location) where the exchange is supposed to take place. Thus the example above may be rewritten as:

$$client\{\!|\, \textbf{role}\ \texttt{auth\_client}.port\_80^{server}\langle index.html\rangle.P \,|\!\}_{\rho} \\ \|\ server\{\!|\, port\_80(x).Q \,|\!\}_{\rho'}$$

We also allow user names to be exchanged during communications. This feature adds flexibility and realism to the language, since in distributed systems users have only a partial and evolving knowledge of their execution environment. For example, the client above can be generalized to leave the server identity unspecified and to dynamically retrieve it with an input from channel *choose_a_server*:

$$client\{\!|\, \textbf{role}\ \texttt{auth\_client}.choose\_a\_server(x) \\ .port\_80^{x}\langle index.html\rangle.P \,|\!\}_{\rho}$$

More details on our calculus, together with some illustrative examples, will be given in Section 1.

The mapping among users, rôles and permissions, which controls the access of subjects to objects, is achieved by a pair of relations $(\mathcal{U};\mathcal{P})$, called *RBAC schema*. In $(\mathcal{U};\mathcal{P})$, relation $\mathcal{U}$ is the association users-to-rôles, while $\mathcal{P}$ is the association permissions-to-rôles. As a first contribution of this paper, we define in Section 2 a type system which complements the dynamics of the calculus by providing static guarantees that systems not respecting a given RBAC schema are rejected. In the client/server example above, a client not authenticated (i.e. interacting with the server without having previously performed a **role** `auth_client`) would be rejected, if the RBAC schema enables only authorized users to perform HTTP requests.

Often, the overall structure of a distributed system cannot be known statically. Thus, a typing approach may not be usable in practice. What is needed is a technique to study system components in isolation, compositionally, and under different schemata. Hence, as a second contribution, in Section 3 we introduce a labeled transition system to give a structured operational semantics to programs, and account for the dynamic checks necessary to enforce RBAC policies. Such labeled transition system yields a bisimulation equivalence, adequate with respect to a standardly defined (typed) barbed congruence, that allows us to prove some interesting algebraic laws. As an example, we show how RBAC schemata may change the semantic theory of the π calculus. Consider the following system, adapted from the client/server example above:

$$(\nu\, port\_80^{server}\!:R)(server\{\!|\, port\_80(x).Q \,|\!\}_{\rho'} \\ \|\ client\{\!|\, port\_80^{server}\langle index.html\rangle.P \,|\!\}_{\emptyset})$$

where $(\nu\, port\_80^{server}:R)$ is the standard restriction operator of a typed π calculus (it declares $port\_80^{server}$ at type R and limits the visibility of the channel to *client* and *server* only). By resuming the assumption that only authorized users can perform HTTP requests, the above system is blocked, i.e. it is equivalent to the empty system $\mathbf{0}$, because the client has not been authenticated. On the contrary, in the π calculus a similar term would have been equivalent to the term resulting from the client/server exchange.

In Section 4 we use types and bisimulations to deal with three meaningful examples: finding the 'minimal' RBAC schema to execute a system, refining a system to be well-typed with respect to a given schema (whenever possible), and minimizing the number of users in a given system without changing the overall behaviour. We conclude by comparing our approach with related work in Section 5. In this extended abstract all proofs are omitted, as is much of the discussion; complete proofs can be found in [3].

CHANNELS:	$a^r, b^s, \ldots \in C = \mathcal{N}_c \times \mathcal{N}_u$
IDENTIFIERS:	$u, v, \ldots \in \mathcal{N}_u \cup \mathcal{V} \cup C \cup (\mathcal{N}_c \times \mathcal{V})$
VALUES:	$m, n, \ldots \in \mathcal{N}_u \cup C$
PROCESSES:	$P, Q ::= \mathbf{nil} \mid P \mid Q \mid\ !P \mid\ [u = v]P$
	$\mid\ (\nu a\!:\!R)P \mid\ a(x).P \mid\ u\langle v\rangle.P$
	$\mid\ \mathbf{role}\, R.P \mid\ \mathbf{yield}\, R.P$
SYSTEMS:	$A, B ::= \mathbf{0} \mid\ r\{\!|\, P \,|\!\}_{\rho} \mid\ A \,\|\, B \mid\ (\nu a^r\!:\!R)A$

Table 1. Syntax of the Calculus

1 The Language

In this section we introduce our calculus formally. First, we define syntax and operational semantics; then, we formalize the RBAC schema to describe the rôles-to-users and permissions-to-rôles assignment.

1.1 Syntax

The calculus is a conservative extension of the π calculus. We assume the following countable and pairwise disjoint sets: $\mathcal{R}$ of rôle names, ranged over by $R, S, \ldots$; $\mathcal{N}_u$ of user names, ranged over by $r, s, \ldots$; $\mathcal{N}_c$ of channel names, ranged over by $a, b, \ldots$; and $\mathcal{V}$ of variables, ranged over by x, y, z. The syntax of the calculus is given in Table 1, with restricted channels decorated with a rôle as described in Section 1.3.

A system consists of the parallel composition of user sessions that can share channels. A user session $r\{\!|\, P \,|\!\}_\rho$ represents a *user* named r executing session P with the set ρ recording r's active rôles. Observe that different sessions of the same user can run in parallel within a system A: this is the usual notion of sessions in RBAC models.

Processes **nil**, $P \mid Q$, $!P$, $[u = v]P$, $(\nu a : R)P$, $a(x).P$, $u\langle v\rangle.P$ are the ordinary π-like constructs representing respectively the inactive process, parallel composition of processes, replication (to model recursive process behaviors), value matching, restriction of channel names and standard input/output actions over channels. (As usual, in the rest of the paper we will omit trailing inactive processes.) The novelty of the calculus resides in the actions **role** R and **yield** R, and in the locality of channels, as already described in the Introduction. Actions **role** R and **yield** R implement activations/deactivations of rôles in the user session they belong to, and modify the session rôles accordingly.

Channels are uniquely associated to users. The set of channels C is formed by coupling a channel name with a user name, and it is ranged over by $a^r, b^s, \ldots$. Identifiers, ranged over by $u, v, \ldots$, denote user names, variables, channels and compound entities made up by a channel name and a variable. The only transmissible *values* are user names and channels and are ranged over by $m, n, \ldots$. Channel names cannot be transmitted, as they make little sense without the indication of the user owning them. Input channels cannot be variables and are not decorated with a user name. This is a syntactic means to localize them, as input channels implicitly belong to the user the appear in. On the other hand, output actions must indicate the name of the user containing the invoked channel. For example, $r\{\!|\, a^s\langle \ldots \rangle.P \,|\!\}_\rho$ models a user r trying to communicate along channel a associated to user s (if any). Notice also that a process like $a(x).b^x\langle v\rangle.P$ can be accepted but, in order to be executed, at run-time x must be assigned a user name r which owns an input channel b^r. These properties will be enforced by the type system of Section 2.

Restrictions $(\nu a : R)P$ and $(\nu a^r : R)A$ and the input prefix $a(x).P$ act as binders for channel name a, channel a^r and variable x, respectively. The sets of free and bound channels in a system A, written $\mathrm{Fc}(A)$ and $\mathrm{Bc}(A)$, are defined accordingly, and so is alpha-conversion. Just notice that $(\nu a : R)P$ within user r binds channel a^r. The formal definition of free and bound channels is in the full paper [3]; here we assume that systems are closed (i.e. with no free variables), that bound channels are pairwise distinct and different from the free ones. Furthermore, observe that user names cannot be restricted. This seems reasonable since the creation of a new user is a sensitive operation: it has to be performed by the system administrator, as it may affect the RBAC policy underlying the entire system.

In this paper, we denote with $\widetilde{\_}$ a possibly empty tuple of entities of kind $\_$. Moreover, we write $\widetilde{a^r} : \widetilde{R}$ to denote the tuple $\{a_1^r : R_1, \ldots, a_k^r : R_k\}$, for $k \geq 0$. Sometimes, we shall use tuples as sets (i.e. without considering the order of their elements) and we write, e.g., $b^s \in \widetilde{a^r}$ or $b^s : S \in \widetilde{a^r} : \widetilde{R}$.

1.2 Dynamic Semantics

The dynamics of the calculus is given in the form of a *reduction relation*. As customary, the reduction semantics is based on an auxiliary relation called *structural congruence*, $\equiv$, which brings the participants of a potential interaction to contiguous positions.

Definition 1.1 (Structural Congruence). The structural congruence relation, $\equiv$, is the least congruence on systems which equates alpha-convertible systems, makes $\|$ and $\mid$ commutative and associative with identities respectively **0** and **nil**, and satisfies the following laws.

$$
\begin{array}{rcll}
r\{\!|\, P \mid Q \,|\!\}_\rho & \equiv & r\{\!|\, P \,|\!\}_\rho \;\|\; r\{\!|\, Q \,|\!\}_\rho & \\
r\{\!|\, (\nu a : R)P \,|\!\}_\rho & \equiv & (\nu a^r : R) r\{\!|\, P \,|\!\}_\rho & \\
(\nu a^r : R)(\nu b^s : S)A & \equiv & (\nu b^s : S)(\nu a^r : R)A & \\
(\nu a^r : R)A \;\|\; B & \equiv & (\nu a^r : R)(A \;\|\; B) & \text{if } a^r \notin \mathrm{Fc}(B) \\
r\{\!|\, !P \,|\!\}_\rho & \equiv & r\{\!|\, P \mid !P \,|\!\}_\rho & \\
r\{\!|\, [u = u]P \,|\!\}_\rho & \equiv & r\{\!|\, P \,|\!\}_\rho &
\end{array}
$$

Definition 1.2 (Reduction Relation). The reduction relation, $\longmapsto$, is the least relation on systems satisfying the following laws.

$$
r\{\!|\, a(x).P \,|\!\}_\rho \;\|\; s\{\!|\, a^r\langle n\rangle.Q \,|\!\}_{\rho'} \longmapsto r\{\!|\, P[n/x] \,|\!\}_\rho \;\|\; s\{\!|\, Q \,|\!\}_{\rho'}
$$

$$
r\{\!|\, \mathbf{role}\, R.P \,|\!\}_\rho \longmapsto r\{\!|\, P \,|\!\}_{\rho \cup \{R\}}
$$

$$
r\{\!|\, \mathbf{yield}\, R.P \,|\!\}_\rho \longmapsto r\{\!|\, P \,|\!\}_{\rho - \{R\}}
$$

$$
\frac{A \longmapsto A'}{A \;\|\; B \longmapsto A' \;\|\; B} \qquad \frac{A \longmapsto A'}{(\nu a^r : R)A \longmapsto (\nu a^r : R)A'}
$$

$$
\frac{A \equiv A' \quad A' \longmapsto B' \quad B' \equiv B}{A \longmapsto B}
$$

All structural rules are standard, but the first two. The first states that a session of user r with rôles ρ hosting two processes running in parallel can be split in two parallel sessions of r with rôles ρ. The second one states that a restriction of a channel name inside a user can be turned into a restriction over the corresponding channel at the system level. Similarly, the reduction relation is an extension of [18] with the rules for actions **role** R and **yield** R. The first action adds R to the rôles ρ activated in the current session, while the second one removes R from ρ. Notice

that, by exploiting the first structural rule and the rules for **role**/**yield**, the user $r\{\!|\, \mathbf{role}\, R.P \mid \mathbf{yield}\, S.Q \,|\!\}_\rho$ evolves into $r\{\!|\, P \,|\!\}_{\rho\cup\{R\}} \parallel r\{\!|\, Q \,|\!\}_{\rho-\{S\}}$, i.e. actions **role**/**yield** only affect the process thread executing them.

1.3 RBAC Schema

To conclude the presentation of the RBAC96 model, we need to define the *RBAC schema*, i.e., the rôles-to-users and permissions-to-rôles associations, where permissions enable the actions a user can perform within a system.

Managing rôles and their interrelationships is a difficult and sensitive task that is often centralized and delegated to a small team of security administrators. In our framework, the RBAC schema consists of a pair of finite relations $(\mathcal{U}; \mathcal{P})$, where $\mathcal{U}$ assigns rôles to users, while $\mathcal{P}$ assigns permissions to rôles. More formally,

$$\mathcal{U} \subseteq_{\mathrm{fin}} (\mathcal{N}_u \cup C) \times \mathcal{R} \qquad \mathcal{P} \subseteq_{\mathrm{fin}} \mathcal{R} \times \mathcal{A}$$

where $\mathcal{A} \triangleq \{R^\uparrow, R^?, R^!\}_{R\in\mathcal{R}}$ represents the set of performable actions. Intuitively, permission $R^\uparrow$ determines the possibility to activate rôle R (via the action **role**), while permissions $R^?$ and $R^!$ determine the possibility of performing input and output actions over a channel of rôle R, respectively. Notice that permissions over input/output actions are not defined in terms of channels, but of channel rôles. In this way, we are flexible enough to model both the permission to communicate over a single channel (when the relation $\mathcal{U}$ maps only one channel to a rôle), and the permission to communicate over the member of a group of channels (when relation $\mathcal{U}$ maps more than one channel to the same rôle). Such a case may be useful in situations where more channels can handle the same kind of requests (cf. Example 1 for a possible situation). Observe that, if $\mathcal{U}$ assigns rôle R to a channel, then the permissions assigned to R by $\mathcal{P}$ are irrelevant; that is, $\mathcal{P}$ matters only for users. Moreover, since channels can be considered as methods provided by users, it seems reasonable that each channel is assigned only one rôle. A RBAC schema satisfying this last requirement is called *well-defined*; in the following we shall only consider well-defined RBAC schemata. Observe that in $\mathcal{A}$ no permission represents actions **yield**. Indeed, we assume that a rôle can be deactivated if (and only if) it has been activated before.

To conclude the presentation of our language, we now give a couple of examples using the features introduced so far. We use the following notational conventions. We use $\_$ as a generic placeholder, and write $\mathcal{U}(\_)$ to denote the set of all rôles R such that $(\_, R) \in \mathcal{U}$; we call the left projection of $\mathcal{U}$ its *domain*, and proceed analogously for $\mathcal{P}$. Finally, we let $\mathcal{P}(\rho)$ mean $\bigcup_{R\in\rho} \mathcal{P}(R)$.

Example 1. Let us now formalize in our framework a scenario where a bank client is waiting to be served by one of the branch cashiers available. There are two users, r and s, representing respectively the client and the bank branch, while cashiers are modeled as channels belonging to user s, named $c_1, \ldots, c_n$. The rôles available are `client` and `cashier`. Relation $\mathcal{U}$ assigns rôle `client` to user r and `cashier` to channels c_i, while $\mathcal{P}$ assigns to `client` the permission to communicate with any of the cashiers, i.e., $(\texttt{client}, \texttt{cashier}^!) \in \mathcal{P}$. In this way, r can indistinctly activate any of the cashier methods. The overall system can be described as follows (where we use Π as a shorthand for parallel composition):

$$\begin{array}{l} r\{\!|\, \mathbf{role}\, \texttt{client}.\mathit{enqueue}^s\langle r\rangle.\mathit{dequeue}(z). \\ \quad z\langle \mathit{req}_1\rangle.\cdots.z\langle \mathit{req}_k\rangle.z\langle \mathit{stop}\rangle.\mathbf{yield}\, \texttt{client} \,|\!\}_\rho \quad \parallel \\ s\{\!|\, (\nu\, \mathit{free} : \texttt{scheduling})(\\ \quad !\mathit{enqueue}(x).\mathit{free}(y).\mathit{dequeue}^x\langle y\rangle \mid \Pi_{i=1}^n \mathit{free}^s\langle c_i^s\rangle \mid \\ \quad \Pi_{i=1}^n\, !c_i(x).(\\ \qquad [x = \mathit{withdrw\_req}]\langle \text{handle withdraw request}\rangle \mid \\ \qquad [x = \mathit{dep\_req}]\langle \text{handle deposit request}\rangle \mid \ldots \mid \\ \qquad [x = \mathit{stop}]\, \mathit{free}^s\langle c_i^s\rangle)\ )\ \ |\!\}_{\rho'} \end{array}$$

Once the client enters the bank (i.e., she activates rôle `client`), she queues up and waits to be served. When one of the cashiers becomes available (information maintained internally by the bank via the reserved channel *free* used for cashiers' scheduling), the client is notified and can make requests along the received channel z. Cashiers repeatedly receive requests; we assume methods to handle money withdraw and deposit (for simplicity, we do not consider the order in which clients arrive; a system of queues can however be added routinely).

Example 2 (Prerequisite rôle). In some circumstances, one may want to require a rôle to be activated only by a user already playing a certain rôle. This is a particular model of constrained RBAC called *prerequisite rôle* (see, e.g., [17]). In the banking scenario of Example 1, imagine that r is member of rôles `client`, `user` and `authenticated_user`, and that the bank policy requires a preliminary authentication phase to identify its clients. This can be implemented by having $(\texttt{authenticated\_user}, \texttt{client}^\uparrow) \in \mathcal{P}$; hence `authenticated_user` must be present in ρ to enable the evolution of r given above.

Example 2 shows that some form of 'default' rôle may be needed to kick-start users' activities. Hence, ρ in $r\{\!|\, P \,|\!\}_\rho$ is used both to record the rôles activated in the session and to assign some default rôles to r at the outset.

2 Static Semantics

The type system described below provides static guarantees that the set of actions performed by any user during the

Typing Identifiers:

$$\text{(T-Id}_1\text{)}\quad \frac{\Gamma(\_) = \rho[\widetilde{a} : \widetilde{C}] \qquad \_ \in \mathcal{N}_u \cup \mathcal{V}}{\Gamma \vdash \_ : \rho[\widetilde{a} : \widetilde{C}]} \qquad \text{(T-Id}_2\text{)}\quad \frac{\Gamma(\_) = \rho[\widetilde{b} : \widetilde{C}, a : C, \widetilde{b'} : \widetilde{C'}] \qquad \_ \in \mathcal{N}_u \cup \mathcal{V}}{\Gamma \vdash a^{\text{-}} : C}$$

Typing Processes:

$$\text{(T-Input)}\quad \frac{\Gamma \vdash a^r : R(T) \quad R^? \in \mathcal{P}(\rho) \quad \Gamma, x : T; \rho \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r a(x).P} \qquad \text{(T-Output)}\quad \frac{\Gamma \vdash u : R(T) \quad \Gamma \vdash v : T \quad R^! \in \mathcal{P}(\rho) \quad \Gamma; \rho \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r u\langle v \rangle.P}$$

$$\text{(T-Rôle)}\quad \frac{\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}] \quad R \in \rho' \quad R^{\uparrow} \in \mathcal{P}(\rho) \quad \Gamma; \rho \cup \{R\} \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r \textbf{role}\, R.P} \qquad \text{(T-Yield)}\quad \frac{R \in \rho \qquad \Gamma; \rho - \{R\} \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r \textbf{yield}\, R.P}$$

$$\text{(T-Nil)}\quad \frac{}{\Gamma; \rho \vdash^{\mathcal{P}}_r \textbf{nil}} \qquad \text{(T-Par)}\quad \frac{\Gamma; \rho \vdash^{\mathcal{P}}_r P \quad \Gamma; \rho \vdash^{\mathcal{P}}_r Q}{\Gamma; \rho \vdash^{\mathcal{P}}_r P \mid Q} \qquad \text{(T-Bang)}\quad \frac{\Gamma; \rho \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r !P}$$

$$\text{(T-Match)}\quad \frac{\Gamma; \rho \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r [u = v]P} \qquad \text{(T-Res)}\quad \frac{\Gamma, a^r : R(T); \rho \vdash^{\mathcal{P}}_r P}{\Gamma; \rho \vdash^{\mathcal{P}}_r (\nu a : R)P}$$

Typing Systems:

$$\text{(T-Empty)}\quad \frac{}{\Gamma \vdash^{\mathcal{P}} \mathbf{0}} \qquad \text{(T-Session)}\quad \frac{\Gamma \vdash r : \rho'[\widetilde{a} : \widetilde{C}] \quad \rho \subseteq \rho' \quad \Gamma; \rho \vdash^{\mathcal{P}}_r P}{\Gamma \vdash^{\mathcal{P}} r[\![\, P \,]\!]_\rho} \qquad \text{(T-SysPar)}\quad \frac{\Gamma \vdash^{\mathcal{P}} A \qquad \Gamma \vdash^{\mathcal{P}} B}{\Gamma \vdash^{\mathcal{P}} A \parallel B} \qquad \text{(T-SysRes)}\quad \frac{\Gamma, a^r : R(T) \vdash^{\mathcal{P}} A}{\Gamma \vdash^{\mathcal{P}} (\nu a^r : R)A}$$

Table 2. Typing Rules

computation respects the RBAC schema, given an initial set ρ of activated rôles. The syntax of types can be defined by the following productions (recall that $\widetilde{\_}$ denotes a possibly empty tuple of entities of kind $\_$)

$$\begin{array}{lll} \textit{Message Types} & T & ::= \quad \rho[\widetilde{a} : \widetilde{C}] \mid C \\ \textit{Channel Types} & C & ::= \quad R(T) \end{array}$$

Type $\rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)]$ can be assigned to a user r belonging to rôles in ρ and owning channels $\widetilde{a^r}$ ordinately of type $\widetilde{R(T)}$. Type $R(T)$ can be assigned to channels exchanging values of type T and belonging to rôle R.

A *typing environment* Γ is a finite partial mapping from $\mathcal{N}_u \cup \mathcal{V}$ into types; thus we write $\Gamma(\_) = T$ to refer to the type T of the user name or variable $\_$. A typing environment can be extended as follows:

$$\Gamma, x : T \triangleq \Gamma \uplus \{x : T\}$$
$$\Gamma, a^r : C \triangleq \Gamma'$$

where

$$\Gamma'(s) = \begin{cases} \Gamma(s) & \text{if } s \neq r \\ \rho[a : C, \widetilde{b} : \widetilde{C}] & \text{if } s = r,\ a \notin \widetilde{b}, \\ & \quad \text{and } \Gamma(r) = \rho[\widetilde{b} : \widetilde{C}] \end{cases}$$

In the rest of the paper, we denote with $\uplus$ the union of functions/relations with disjoint domains. A typing environment Γ can be used to type a system under a schema $(\mathcal{U}; \mathcal{P})$ only if the rôle information in Γ respects the associations in $\mathcal{U}$. This intuition is formalized by the following definition.

Definition 2.1. Given a RBAC schema $(\mathcal{U}; \mathcal{P})$ and a typing environment Γ, we say that Γ *respects* $\mathcal{U}$ if, for all $r \in dom(\Gamma)$ with $\Gamma(r) = \rho[a_1 : R_1(T_1), \ldots, a_n : R_n(T_n)]$, it holds that $\mathcal{U}(r) = \rho$ and $\mathcal{U}(a_i^r) = \{R_i\}$, for all $i = 1, \ldots, n$.

The primary judgments of the type system are of the form $\Gamma \vdash^{\mathcal{P}} A$, that should be read as "the system A is well-formed with respect to environment Γ and relation $\mathcal{P}$". This fact, together with the requirement that Γ respects $\mathcal{U}$, implies that A respects the RBAC schema $(\mathcal{U}; \mathcal{P})$. To infer the main judgment, we rely on two auxiliary judgments, one for identifiers and one for processes. Judgment $\Gamma \vdash u : T$ states that the identifier u has type T in Γ; judgment $\Gamma; \rho \vdash^{\mathcal{P}}_r P$ states that P respects Γ and $\mathcal{P}$ when it is run in a session of r with rôles ρ activated.

The typing rules are collected in Table 2. Most of them are self-explanatory; we comment below the most significant ones, i.e. those related to the actions in our calculus. The underlying idea beyond these rules is that an action can

be executed only if the current session has activated a rôle enabling the action. Rule (T-Input) states that, for typing $a(x).P$ in a session of r where rôles ρ are activated, we need to establish that a^r has type $R(T)$ in Γ, that inputs over a channel of group R can be performed when playing rôles ρ and that P is typeable once assumed that x has type T. Rule (T-Output) is similar: it checks that an output over a channel of group R is allowed when rôles in ρ are activated. Moreover, it also requires that the transmitted value v can be assigned type T in Γ. Rule (T-Rôle) states that for typing process $\mathbf{role}\,R.P$ in a session of r where rôles ρ are activated, we need to check that r can assume rôle R, that ρ enables the activation and that P is typeable for r having activated $\rho \cup \{R\}$. Rule (T-Yield) states that process $\mathbf{yield}\,R.P$ is legal for r only when R has been previously activated and if P is typeable for r when R is off.

Finally, notice that in rules (T-Res) and (T-SysRes) the type of the restricted channel is not tracked in the restriction construct. Indeed, for typechecking purposes, it suffices to ensure that the new channel is used coherently by all the processes accessing it. To this aim, we only need to invent a suitable T when applying the rules and verify that all the accesses to the channel conform to T.

Definition 2.2 (Well-typedness). Given a RBAC schema $(\mathcal{U};\mathcal{P})$ and a system A, we say that A is *well-typed for* $(\mathcal{U};\mathcal{P})$ if there exists a typing environment Γ respecting $\mathcal{U}$ such that $\Gamma \vdash^{\mathcal{P}} A$.

We now prove the soundness of the type system in the standard way, i.e., by proving subject reduction and type safety, which ensure that only systems abiding by the RBAC schema are allowed (i.e., users performing actions permitted by their duly activated rôles).

Theorem 2.1 (Subject Reduction). *If $\Gamma \vdash^{\mathcal{P}} A$ and $A \longmapsto A'$, then $\Gamma \vdash^{\mathcal{P}} A'$.*

Theorem 2.2 (Type Safety). *Let A be a well-typed system for $(\mathcal{U};\mathcal{P})$. Then*

1. *whenever $A \equiv (\nu\, \widetilde{a^r}:\widetilde{R})(A' \parallel r\{\!|\, P \,|\!\}_\rho)$, it holds that $\rho \subseteq \mathcal{U}(r)$*
2. *whenever $A \equiv (\nu\, \widetilde{a^r}:\widetilde{R})(A' \parallel r\{\!|\, \mathbf{role}\,R.P \,|\!\}_\rho)$, it holds that $R \in \mathcal{U}(r)$ and $R^{\uparrow} \in \mathcal{P}(\rho)$*
3. *whenever $A \equiv (\nu\, \widetilde{a^r}:\widetilde{R})(A' \parallel r\{\!|\, \mathbf{yield}\,R.P \,|\!\}_\rho)$, it holds that $R \in \rho$*
4. *whenever $A \equiv (\nu\, \widetilde{a^r}:\widetilde{R})(A' \parallel r\{\!|\, b(x).P \,|\!\}_\rho)$, it holds that either $b^r : S \in \widetilde{a^r}:\widetilde{R}$ and $S^? \in \mathcal{P}(\rho)$, or $b^r \notin \widetilde{a^r}$ and $S^? \in \mathcal{P}(\rho)$, where $\{S\} = \mathcal{U}(b^r)$*
5. *whenever $A \equiv (\nu\, \widetilde{a^r}:\widetilde{R})(A' \parallel r\{\!|\, b^s\langle n\rangle.P \,|\!\}_\rho)$, it holds that either $b^s : S \in \widetilde{a^r}:\widetilde{R}$ and $S^! \in \mathcal{P}(\rho)$, or $b^s \notin \widetilde{a^r}$ and $S^! \in \mathcal{P}(\rho)$, where $\{S\} = \mathcal{U}(b^s)$*

Example 3. Let us consider again the banking scenario described in Example 1. To illustrate the type system introduced above, let us give a possible typing for the system. Let $T_{csh} \triangleq \texttt{cashier}(\{\texttt{request}\}[])$ be the type of the cashiers, i.e., channels belonging to rôle `cashier` and exchanging values of type $\{\texttt{request}\}[]$. Type $\{\texttt{request}\}[]$ represents the possible requests of clients; syntactically, values of this type are names of users belonging to rôle `request` which do not provide any channel. Moreover, we let

$$T_{cl} \triangleq (\rho \cup \{\texttt{client}\})[\mathit{dequeue} : \texttt{cashier\_get}(T_{csh})]$$

be the type of r. This represents users belonging to rôles in $\rho \cup \{\texttt{client}\}$ and owning a channel named *dequeue* of type $\texttt{cashier\_get}(T_{csh})$. Then, a suitable typing environment Γ is

$$\begin{aligned} r &\mapsto T_{cl} \\ s &\mapsto \rho'[\mathit{enqueue} : \texttt{cashier\_req}(T_{cl}), \\ &\qquad c_1 : T_{csh}, \ldots, c_n : T_{csh}] \\ \mathit{withdrw\_req} &\mapsto \{\texttt{request}\}[] \\ \mathit{dep\_req} &\mapsto \{\texttt{request}\}[] \\ \ldots &\mapsto \ldots \\ \mathit{stop} &\mapsto \{\texttt{request}\}[]. \end{aligned}$$

A suitable permissions-to-rôles assignment $\mathcal{P}$ is

$$\begin{aligned} &\{\texttt{cashier\_req}^!, \\ &\quad \texttt{cashier\_get}^?, \texttt{cashier}^!\} \subseteq \mathcal{P}(\texttt{client}); \\ &\qquad\qquad \{\texttt{client}^{\uparrow}\} \subseteq \mathcal{P}(\rho); \\ A \cup &\{\texttt{cashier\_req}^?, \\ &\quad \texttt{scheduling}^?, \texttt{scheduling}^!, \\ &\quad \texttt{cashier\_get}^!, \texttt{cashier}^?\} \subseteq \mathcal{P}(\rho'); \end{aligned}$$

where $A \subseteq \mathcal{A}$ is a set of action permissions that allow the handling of client's requests. The system of Example 1 is well-typed for any schema $(\mathcal{U},\mathcal{P})$ such that Γ respects $\mathcal{U}$.

Example 4. In the real world, it is unrealistic to allow any bank client to ask for any kind of bank operation. For instance, when a client applies for credit, she is always asked for some credentials. To model this finer scenario, we let each available operation to be modeled as a specific method, which can be activated through a specific channel (e.g., channel *wdrw* handles withdraw requests, *opn* handles open account requests, *cc* handles credit card requests, etc.). The communication along different channels requires different rôles and, thus, it is a way to control the credentials of the client. In this setting, the cashier c_i of Example 1 is implemented by the following process (the remaining behaviour

of the bank is implemented as in Example 1):

$$\begin{aligned} c_i(x).(\; & [x = \mathit{withdrw\_req}]\, \mathit{wdrw}(y).\ldots \mid \\ & [x = \mathit{open\_req}]\, \mathit{opn}(y).\ldots \mid \\ & [x = \mathit{creditcard\_req}]\, \mathit{cc}(y).\ldots \mid \ldots \mid \\ & [x = \mathit{stop}]\, \mathit{free}^s\langle c_i^s \rangle \;) \end{aligned}$$

Let relation $\mathcal{U}$ assign channel *wdrw* (respectively, *opn* and *cc*) the group `wdrw` (respectively, `opn` and `cc`), and $\mathcal{P} = \{$ $(\texttt{rich\_client}, \texttt{cc}^!)$, $(\texttt{client}, \texttt{wdrw}^!)$, $(\texttt{user}, \texttt{opn}^!)$, $(\texttt{user}, \texttt{client}^\uparrow)$, $(\texttt{rich}, \texttt{rich\_client}^\uparrow)$ $\}$. Thus, under this schema, the client

$$r\{\!|\, \mathbf{role}\ \texttt{client}.\mathit{enqueue}^s\langle r\rangle.\mathit{dequeue}(z).z\langle \mathit{creditcard\_req}\rangle. \\ \mathit{cc}^s\langle \mathit{signature}\rangle.z\langle \mathit{stop}\rangle.\mathbf{yield}\ \texttt{client}\, |\!\}_{\{\texttt{user}\}}$$

is not well-typed because she has not activated the correct rôle for performing credit card requests. Indeed, the type-checking fails when applying the rule (T-Output) to action $\mathit{cc}^s\langle \mathit{signature}\rangle$ because $\texttt{cc}^! \notin \mathcal{P}(\{\texttt{user}, \texttt{client}\})$. On the other hand, the following clients do type-check:

$$r_1\{\!|\, \mathbf{role}\ \texttt{rich\_client}.\mathit{enqueue}^s\langle r\rangle.\mathit{dequeue}(z). \\ z\langle \mathit{creditcard\_req}\rangle.\mathit{cc}^s\langle \mathit{signature}\rangle.z\langle \mathit{stop}\rangle\, |\!\}_{\{\texttt{rich}\}}\ ;$$

$$r_2\{\!|\, \mathbf{role}\ \texttt{client}.\mathit{enqueue}^s\langle r\rangle.\mathit{dequeue}(z).z\langle \mathit{withdrw\_req}\rangle. \\ \mathit{wdrw}^s\langle \mathit{sum}\rangle.z\langle \mathit{stop}\rangle\, |\!\}_{\{\texttt{user}\}}\ ;$$

$$r_3\{\!|\, \mathit{enqueue}^s\langle r\rangle.\mathit{dequeue}(z).z\langle \mathit{open\_req}\rangle. \\ \mathit{opn}^s\langle \mathit{personal\_data}\rangle.z\langle \mathit{stop}\rangle\, |\!\}_{\{\texttt{user}\}}\ .$$

We conclude this section remarking that our type system is not powerful enough to type all legal systems. For example, the absence of a recursive type constructor makes the system $r\{\!|\, a^r\langle r\rangle\, |\!\}_\rho$ untypeable. Recursive types can be standardly handled as in [15]. Similarly, we have no notion of subtyping. Thus, a channel must always carry values exactly of the same type. By introducing standard π calculus subtyping (see e.g. [16, 18]), a more liberal typing discipline can be developed in a standard way. For the sake of simplicity, here we preferred to focus on the core characteristics needed in our setting.

3 Observational Semantics

Often, the overall structure of a distributed system cannot be known statically. Thus, the typing approach described in the previous section, even if interesting from a theoretical point of view, may not be usable in practice. In this section, we introduce a labeled transition system (LTS, for short) which embodies dynamic policy checks, and allows us to study (not necessarily well-typed) system components in isolation and compositionally. The LTS also provides a tight operational model for the minimal engine underlying any implementation of a RBAC-based run-time system. We define a standard bisimulation over the LTS and show that it is adequate with a typed barbed congruence, a relevant result in at least two respects. Firstly, it signifies that our bisimulation is a sensible equivalence to consider, as it agrees with the (typed) contextual semantics derived from an elementary, natural class of observables. Secondly, it provides us with a powerful co-inductive proof technique for barbed congruence.

The standard way to describe the interactions a system can offer externally is by labeling the system evolution with this information. Thus, we define a labeled transition system, $\xrightarrow{\mu}$, that makes apparent the action performed (and, thus, the external interaction offered). Since we do not require $\xrightarrow{\mu}$ to act only on well-typed terms, the LTS comes equipped with runtime checks with respect to the RBAC schema considered to block the execution of illegal actions.

The LTS evolves from the π calculus' early-style transition system. In order to account for systems' rôles varying over time, the LTS relates *configurations*, i.e. pairs $(\mathcal{U};\mathcal{P}) \triangleright A$ made up of a RBAC schema $(\mathcal{U};\mathcal{P})$ and a system A. Configurations are ranged over by D, E. The labels of the LTS are derived from those of the π calculus and can be described as follows.

$$\mu \quad ::= \quad \tau \;\mid\; a^r n \;\mid\; a^r n : R \;\mid\; \overline{a}^r n \;\mid\; \overline{a}^r n : R$$

Label τ represents an internal computation of the system. Labels $\overline{a}^r n$ and $a^r n$ describe the intention to send/receive value n, known to the environment, on/from channel a^r. Labels $\overline{a}^r n : R$ and $a^r n : R$ are similar to but the value sent/received is 'fresh' (i.e. unknown to the environment) and has group R. Functions $\mathrm{Fc}(\_)$ and $\mathrm{Bc}(\_)$ are easily extended to labels. In particular, b^s is the bound channel of μ whenever μ is either $a^r b^s : R$ or $\overline{a}^r b^s : R$; free channels are defined accordingly.

The rules defining $\xrightarrow{\mu}$ are given in Table 3. The overall structure of the system is similar to π calculus' (see, e.g., [18]). We use rules (LTS-Split), (LTS-Ext), (LTS-Match), (LTS-Bang) and the symmetric versions of rules (LTS-Comm), (LTS-Par) and (LTS-Close) to avoid structural congruence; however, we still implicitly assume alpha-conversion. The premises of rules (LTS-K-Input), (LTS-F-Input), (LTS-Output), (LTS-Rôle) and (LTS-Yield) adapt respectively the premises of the typing rules (T-Input), (T-Output), (T-Rôle) and (T-Yield), and block the evolution of ill-typed systems. Rule (LTS-K-Input) can be applied when the received value is known to the schema, while (LTS-F-Input) is used when a fresh value (i.e. unknown to the schema) is received. In this case, the schema is extended to record the group of the fresh value. Similarly, when extruding a restricted channel b^s, rule (LTS-Open) enlarges the relation $\mathcal{U}$ of the current configuration by recording that b^s has the rôle declared in the restriction. The information

(LTS-Rôle)
$$\frac{R \in \mathcal{U}(r) \qquad R^{\uparrow} \in \mathcal{P}(\rho)}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, \mathbf{role}\, R.P \,|\!\}_{\rho} \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \,|\!\}_{\rho \cup \{R\}}}$$

(LTS-Yield)
$$\frac{R \in \rho}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, \mathbf{yield}\, R.P \,|\!\}_{\rho} \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \,|\!\}_{\rho - \{R\}}}$$

(LTS-K-Input)
$$\frac{\mathcal{U}(a^r) = \{R\} \quad R^{?} \in \mathcal{P}(\rho) \quad n \in dom(\mathcal{U})}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, a(x).P \,|\!\}_{\rho} \xrightarrow{a^r n} (\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P[n/x] \,|\!\}_{\rho}}$$

(LTS-Output)
$$\frac{\mathcal{U}(a^s) = \{R\} \qquad R^{!} \in \mathcal{P}(\rho)}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, a^s\langle n\rangle.P \,|\!\}_{\rho} \xrightarrow{\overline{a}^s n} (\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \,|\!\}_{\rho}}$$

(LTS-Comm)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright A \xrightarrow{a^r n} (\mathcal{U};\mathcal{P}) \triangleright A' \qquad (\mathcal{U};\mathcal{P}) \triangleright B \xrightarrow{\overline{a}^r n} (\mathcal{U};\mathcal{P}) \triangleright B'}{(\mathcal{U};\mathcal{P}) \triangleright A \parallel B \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright A' \parallel B'}$$

(LTS-F-Input)
$$\frac{\mathcal{U}(a^r) = \{R\} \qquad R^{?} \in \mathcal{P}(\rho) \qquad n \notin dom(\mathcal{U})}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, a(x).P \,|\!\}_{\rho} \xrightarrow{a^r n:S} (\mathcal{U} \uplus \{n : S\};\mathcal{P}) \triangleright r\{\!|\, P[n/x] \,|\!\}_{\rho}}$$

(LTS-Open)
$$\frac{(\mathcal{U} \uplus \{b^s{:}S\};\mathcal{P}) \triangleright A \xrightarrow{\overline{a}^r b^s} (\mathcal{U} \uplus \{b^s{:}S\};\mathcal{P}) \triangleright A' \qquad a^r \neq b^s}{(\mathcal{U};\mathcal{P}) \triangleright (\nu b^s{:}S)A \xrightarrow{\overline{a}^r b^s:S} (\mathcal{U} \uplus \{b^s{:}S\};\mathcal{P}) \triangleright A'}$$

(LTS-Close)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright A \xrightarrow{a^r b^s:S} (\mathcal{U}';\mathcal{P}) \triangleright A' \qquad (\mathcal{U};\mathcal{P}) \triangleright B \xrightarrow{\overline{a}^r b^s:S} (\mathcal{U}';\mathcal{P}) \triangleright B' \qquad b^s \notin \mathrm{Fc}(A)}{(\mathcal{U};\mathcal{P}) \triangleright A \parallel B \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright (\nu b^s{:}S)(A' \parallel B')}$$

(LTS-Res)
$$\frac{(\mathcal{U} \uplus \{a^r{:}R\};\mathcal{P}) \triangleright A \xrightarrow{\mu} (\mathcal{U}' \uplus \{a^r{:}R\};\mathcal{P}) \triangleright A' \qquad a^r \notin \mathrm{Fc}(\mu)}{(\mathcal{U};\mathcal{P}) \triangleright (\nu a^r{:}R)A \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright (\nu a^r{:}R)A'}$$

(LTS-Par)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright A \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A' \qquad \mathrm{Bc}(\mu) \cap \mathrm{Fc}(B) = \emptyset}{(\mathcal{U};\mathcal{P}) \triangleright A \parallel B \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A' \parallel B}$$

(LTS-Ext)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright (\nu a^r{:}R)r\{\!|\, P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, (\nu a{:}R)P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}$$

(LTS-Split)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \,|\!\}_{\rho} \parallel r\{\!|\, Q \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \mid Q \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}$$

(LTS-Match)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, [u = u]P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}$$

(LTS-Bang)
$$\frac{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, P \mid !P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}{(\mathcal{U};\mathcal{P}) \triangleright r\{\!|\, !P \,|\!\}_{\rho} \xrightarrow{\mu} (\mathcal{U}';\mathcal{P}) \triangleright A}$$

plus the symmetric version of rules of (LTS-Par), (LTS-Comm) and (LTS-Close)

Table 3. A Labeled Transition System

about a fresh/extruded channel is deleted from the schema when the channel is communicated: indeed, the restriction is pushed back in the system and closes the scope of the channel – cf. rule (LTS-Close). Notice that a bound output can synchronize only with a fresh input (and vice versa), and the rôle declared for the extruded/fresh channel must be the same. Also observe that τ-moves do not modify the schema $(\mathcal{U};\mathcal{P})$.

The semantics given in Definition 1.2 and the LTS just presented are related by the following

Proposition 3.1. *If* $(\mathcal{U};\mathcal{P}) \triangleright A \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright A'$, *we have* $A \longmapsto A'$. *Also, if* A *is well-typed for* $(\mathcal{U};\mathcal{P})$, *then* $A \longmapsto A'$ *implies* $(\mathcal{U};\mathcal{P}) \triangleright A \xrightarrow{\tau} (\mathcal{U};\mathcal{P}) \triangleright B$, *for some* $B \equiv A'$.

Next, we build upon this LTS a standard bisimulation. As usual, $\Longrightarrow$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\mu}{\Longrightarrow}$ denotes $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$. Finally, $\overset{\hat{\mu}}{\Longrightarrow}$ is $\Longrightarrow$ if $\mu = \tau$, and $\overset{\mu}{\Longrightarrow}$ otherwise.

Definition 3.1 (Bisimilarity). A *bisimulation* is a binary symmetric relation $\mathcal{S}$ between configurations such that, if $(D,E) \in \mathcal{S}$ and $D \xrightarrow{\mu} D'$, there exists a configuration E' such that $E \overset{\hat{\mu}}{\Longrightarrow} E'$ and $(D',E') \in \mathcal{S}$. *Bisimilarity*, $\approx$, is the largest bisimulation.

As often happens in typed calculi, $\approx$ is *not* a congruence for all system contexts. Indeed, due to the checks in the LTS for schema compliance, the application of ill-typed contexts can break equivalences. Consider for instance $A \triangleq r\{\!|\, a(x).b^r\langle\cdot\rangle \mid a^r\langle\cdot\rangle \,|\!\}_\rho$ and $B \triangleq \mathbf{0}$, when $a^r \notin dom(\mathcal{U})$, $\mathcal{U}(b^r) = \{R\}$ and $R^?, R^! \in \mathcal{P}(\rho)$. Then $(\mathcal{U};\mathcal{P}) \triangleright A \approx (\mathcal{U};\mathcal{P}) \triangleright B$ but $(\mathcal{U};\mathcal{P}) \triangleright (\nu a^r\!:\!R)A \not\approx (\mathcal{U};\mathcal{P}) \triangleright (\nu a^r\!:\!R)B$. A similar problem arises if $\mathcal{U}(a^r) = \{S\}$ but $S^?, S^! \notin \mathcal{P}(\rho)$. Moreover, some care must be paid when the configurations equated rely on different schemata. Indeed, it is easy to find a situation where $(\mathcal{U}_1;\mathcal{P}_1) \triangleright A_1 \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright A_2$ but $(\mathcal{U}_1;\mathcal{P}_1) \triangleright A_1 \parallel B \not\approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright A_2 \parallel B$: it suffices to find a system B with an action enabled by $\mathcal{P}_1$ but disabled by $\mathcal{P}_2$.

Theorem 3.2 (Congruence Properties of $\approx$). *The following facts hold:*

1. *if* $(\mathcal{U}_1;\mathcal{P}_1) \triangleright A_1 \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright A_2$ *and* $(\mathcal{U}_1;\mathcal{P}_1) \triangleright B \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright B$, *then* $(\mathcal{U}_1;\mathcal{P}_1) \triangleright A_1 \parallel B \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright A_2 \parallel B$;
2. *if* $(\mathcal{U}_1 \uplus \{a^r\!:\!R\};\mathcal{P}_1) \triangleright A_1 \approx (\mathcal{U}_2 \uplus \{a^r\!:\!R\};\mathcal{P}_2) \triangleright A_2$, *then* $(\mathcal{U}_1;\mathcal{P}_1) \triangleright (\nu a^r\!:\!R)A_1 \approx (\mathcal{U}_2;\mathcal{P}_2) \triangleright (\nu a^r\!:\!R)A_2$.

Bisimulation is a sound semantic equivalence, in the sense that it produces no unreasonable equations. To substantiate this claim, we prove its adequacy for a standardly defined typed observational congruence, viz. the *reduction barbed congruence* [11]. This is a touchstone equivalence defined in terms of the reduction relation and of a notion of observability, and then closed under all possible system contexts. The reason to consider a typed congruence is that only well-typed contexts guarantee a reduction behaviour abiding by the RBAC policy. Indeed, the reduction relation performs none of the legality checks hard-coded in the LTS. Hence, the right framework for comparison of $\approx$ and a barbed congruence is a typed one.

In its typed version, barbed congruence is tagged with an environment Γ and a permissions-to-rôles assignment $\mathcal{P}$, to signify that its equations are typeable under Γ and $\mathcal{P}$. Moreover, only contexts typeable under Γ and $\mathcal{P}$ are considered in the definition of congruence. Thus, following the style of [9], we write $\Gamma \models^{\mathcal{P}} A_1 \cong A_2$ to mean that $\Gamma \vdash^{\mathcal{P}} A_i$ for $i = 1,2$ and that A_1 and A_2 exhibit the same behaviour in all environment 'compatible' with Γ and $\mathcal{P}$.

Definition 3.2 (Barbs). The *observation predicate* $A \downarrow \eta$ holds if either $\eta = a^r$ and $A \equiv (\nu\, \widetilde{b^s}\!:\!\widetilde{R})(A' \parallel r\{\!|\, a(x).P \,|\!\}_\rho)$ for $a^r \notin \widetilde{b^s}$, or $\eta = \overline{a^r}$ and $A \equiv (\nu\, \widetilde{b^s}\!:\!\widetilde{R})(A' \parallel s'\{\!|\, a^r\langle n\rangle.P \,|\!\}_\rho)$ for $a^r \notin \widetilde{b^s}$. The predicate $A \Downarrow \eta$ holds if there exists $A \Rightarrow A'$ such that $A' \downarrow \eta$.

Definition 3.3 (Reduction Barbed Congruence). *Reduction barbed congruence* is the largest binary and symmetric typed relation over systems such that the following properties hold whenever $\Gamma \models^{\mathcal{P}} A_1 \cong A_2$.

1. *Barb Preserving:* if $A_1 \downarrow \eta$, then $A_2 \Downarrow \eta$
2. *Reduction Closed:* if $A_1 \longmapsto A_1'$, then there exists a system A_2' such that $A_2 \Rightarrow A_2'$ and $\Gamma \models^{\mathcal{P}} A_1' \cong A_2'$
3. *Contextual:*
   (a) for all $\mathcal{P}'$ and $\widetilde{u}\!:\!\widetilde{T}$ such that $\Gamma, \widetilde{u}\!:\!\widetilde{T}$ is defined, it holds that $\Gamma, \widetilde{u}\!:\!\widetilde{T} \models^{\mathcal{P}\cup\mathcal{P}'} A_1 \cong A_2$
   (b) for all systems B such that $\Gamma \vdash^{\mathcal{P}} B$ it holds that $\Gamma \models^{\mathcal{P}} A_1 \parallel B \cong A_2 \parallel B$;
   (c) for all $a^r : R(T)$ such that $\Gamma = \Gamma', a^r : R(T)$, it holds that $\Gamma' \models^{\mathcal{P}} (\nu a^r\!:\!R)A_1 \cong (\nu a^r\!:\!R)A_2$.

Before comparing $\approx$ and $\cong$, we remark that the chosen barbs only express the ability to interact over channels. Indeed, observing rôle activations/deactivations is not reasonable, as no context can determine whether a user performs a **role**/**yield**: these operations only affect the thread performing them.

The fact that $\approx$ approximates $\cong$ only holds for well-typed configurations, i.e. configurations $(\mathcal{U};\mathcal{P}) \triangleright A$ such that A is well-typed for $(\mathcal{U};\mathcal{P})$. Given a typing environment Γ, we let $\mathcal{U}_\Gamma$ be the rôles-to-users assignment extracted from Γ, that is the least assignment such that, for any association $r : \rho[\widetilde{a} : \widetilde{R(T)}]$ in Γ, it holds that $\mathcal{U}_\Gamma(r) = \rho$ and $\mathcal{U}_\Gamma(a^r) = \{R\}$ for any $a : R(T) \in \widetilde{a} : \widetilde{R(T)}$.

Theorem 3.3 (Soundness of $\approx$). *Let* $\Gamma \vdash^{\mathcal{P}} A$ *and* $\Gamma \vdash^{\mathcal{P}} B$. *If* $(\mathcal{U}_\Gamma;\mathcal{P}) \triangleright A \approx (\mathcal{U}_\Gamma;\mathcal{P}) \triangleright B$ *then* $\Gamma \models^{\mathcal{P}} A \cong B$.

Theorem 3.3 shows that $\approx$ is a sound proof-technique for barbed congruence. However, while the former is relatively easy to use, the latter is very hard to handle because of the contextual closure requirement. We leave as a future work the development of finer techniques (as e.g. in [8, 14]) to prove the converse of Theorem 3.3, i.e. that bisimilarity is complete for barbed congruence.

To conclude, we now list some algebraic laws that illustrate the impact of RBAC on the π calculus. In what follows, we fix a RBAC schema $(\mathcal{U};\mathcal{P})$. The first equation states that a terminated session of a user does not affect the evolution of a system. Indeed, it holds that

$$r\{\!|\,\mathbf{nil}\,|\!\}_\rho \approx \mathbf{0}.$$

This is different from some distributed calculi, like e.g. the Ambient calculus [5], where the presence of a user is relevant. Moreover, by letting α to range over action prefixes (i.e. inputs/outputs and **role**/**yield**), it holds that

$$r\{\!|\,\alpha.P\,|\!\}_\rho \approx \mathbf{0}$$

whenever α is not legal for a session $r\{\!|\cdot|\!\}_\rho$ with respect to the RBAC schema, that is if the premises of rules (LTS-Rôle), (LTS-Yield), (LTS-K-Input), (LTS-F-Input) and (LTS-Output) are not satisfied. This law stresses that LTS and types both enforce the same requirements (compare the runtime checks of the LTS with Theorem 2.2). As a consequence, the following law differentiates our language from the π calculus. Indeed, it holds that

$$(\nu a^r\!:\!R)(r\{\!|\,a(x).P\,|\!\}_\rho \parallel s\{\!|\,a^r\langle n\rangle.Q\,|\!\}_{\rho'}) \approx \mathbf{0},$$

if and only if $R^? \notin \mathcal{P}(\rho)$ or $R^! \notin \mathcal{P}(\rho')$.

Differently from several distributed languages, the user performing an output action is irrelevant. The only relevant aspect is the set of permissions activated when performing the action. This is summarized in the following law:

$$r\{\!|\,b^s\langle n\rangle.\mathbf{nil}\,|\!\}_\rho \approx t\{\!|\,b^s\langle n\rangle.\mathbf{nil}\,|\!\}_\rho.$$

A similar law holds for the **yield** action. On the contrary, relocating an input action usually breaks the equivalence between processes. In particular, we have

$$r\{\!|\,a(x).P\,|\!\}_\rho \not\approx t\{\!|\,a(x).P\,|\!\}_\rho$$

unless both input actions are disabled (in which case both systems are equivalent to **0**). Similarly, it is possible to migrate a **role** R prefix between two users only when R is assigned to both or to none of them. By exploiting these observations, we can find a relocation procedure to minimize the number of users in a system, while maintaining the system overall behaviour, as it will be described in the next section.

4 Applications

In this section, we exploit the theory developed so far in three non-trivial applications of the RBAC model. The first deals with the problem of finding the 'minimal' schema which makes a given system legal. The second is somehow symmetric: given a schema and a system, we aim at arranging **role**/**yield** operations within the system so that the resulting system can be executed with respect to the given schema (if possible). Finally, we give a simple but efficient procedure to determine whether a process can be executed by different users without compromising the functionality of the system. This can be useful to minimize the number of users in a system, while maintaining the overall system behaviour.

Minimal Schema. Let A be a system well-typed for a RBAC schema $(\mathcal{U};\mathcal{P})$. Potentially, there are infinitely many schemata under which the system can run correctly; thus, it seems reasonable to look for a 'minimal' such. This would make the execution/verification of systems more efficient both in space and time: the storage of the schema is reduced in size, thus any query to the schema can be replied faster.

We define the set of configurations for A as $CONF_A = \{(\mathcal{U}';\mathcal{P}') \triangleright A \mid (\mathcal{U}';\mathcal{P}') \text{ is a RBAC schema}\}$. Even if potentially infinite, the relevant part of $CONF_A$ can be effectively built up by considering only the rôles and identifiers occurring in the system A. We now partition $CONF_A$ with respect to $\approx$ and consider the equivalence class containing $(\mathcal{U};\mathcal{P}) \triangleright A$, called $CONF_A^{(\mathcal{U};\mathcal{P})}$. By fixing a metrics over schemata, the minimal schema to run the system A will be a minimal element of $CONF_A^{(\mathcal{U};\mathcal{P})}$.

Clearly, the existence of such a minimal element and the way in which it is chosen depend on a chosen *metrics*. For example, one can consider as a good metrics the value $|\mathcal{U}| + |\mathcal{P}|$, i.e., the size of the schema expressed in terms of the number of couples forming the relations $\mathcal{U}$ and $\mathcal{P}$. In this case, a minimal schema always exists. Other metrics could be based on the number of rôles used to define the schema, on the weight of the permissions associated to some users (once assumed a weight function to discriminate powerful permissions from common ones), on the average number of permissions associated to each rôle, and so on.

Notice that in general A may *not* be well-typed under the minimal schema. This is because the *static* typing procedure over-approximates the behaviour of a system (e.g., it also considers unreachable code and connot type all legal systems, as described at the end of Section 2). The bisimulation-based approach presented here is more accurate since it only considers the effective behaviour of the system. Thus, the schema obtained in this way describes the minimal requirements a schema should satisfy to *run* (and not to type) system A, while maintaining the behaviour of A

under schema $(\mathcal{U};\mathcal{P})$.

Refining Systems to make them Executable. Usually, the task of properly putting **role**/**yield** operations within a system is tedious and error-prone; moreover, it assumes a full knowledge of the RBAC schema at programming time. We now describe a way to add rôle activations/deactivations within a system in such a way that the resulting system can be executed under a given schema, whenever possible. Notice that, given a RBAC schema $(\mathcal{U};\mathcal{P})$ and a system A without actions **role**/**yield**, we can simply refine A in a system A' by activating at the beginning of each session of a (generic) user r all the rôles in $\mathcal{U}(r)$. Intuitively, A' contains all the legal behaviors of A with respect to the RBAC schema given. However, the fact that all the rôles assigned to a user are always activated violates a basilar RBAC design principle: a rôle should be active only when needed. In the following, we give a procedure to refine this approach and obtain a system closer to the RBAC design principles.

Let $\vec{R}$ denote a possibly empty, ordered sequence of rôles and $\mathbf{role}\,\vec{R}$ (resp. $\mathbf{yield}\,\vec{R}$) denote $\mathbf{role}\,R_1.\cdots.\mathbf{role}\,R_n$ (resp. $\mathbf{yield}\,R_1.\cdots.\mathbf{yield}\,R_n$) whenever $\vec{R} = R_1,\cdots,R_n$. The refining procedure replaces any input/output prefix α occurring in session $r\{\!|\cdots|\!\}_\rho$ with the sequence of prefixes $\mathbf{role}\,\vec{R}.\alpha.\mathbf{yield}\,\vec{R}$ where $\vec{R}$ is formed by rôles that r can activate when holding ρ, and that enable the execution of α.[1] Moreover, since there are in principle several such $\vec{R}$, we choose one of the shortests, i.e. a sequence containing the minimum number of elements. Let $en_a(\Gamma,\mathcal{P},r,\rho)$ denote a function which returns a shortest sequence of rôles $\vec{R}$ such that $\Gamma;\rho \vdash_r^{\mathcal{P}} \mathbf{role}\,\vec{R}.a(x).\mathbf{nil}$, and is undefined if no such sequence exists. Function $en_{\bar{u}}(\Gamma,\mathcal{P},r,\rho)$ is similar, but for outputs over u. The refining procedure adapts the type system presented in Section 2; as an example, the following rule adapts (T-Input).

$$\frac{\Gamma \vdash a^r : R(T) \quad en_a(\Gamma,\mathcal{P},\rho,r) = \vec{S} \quad \Gamma, x:T;\rho \vdash_r^{\mathcal{P}} P \gg P'}{\Gamma;\rho \vdash_r^{\mathcal{P}} a(x).P \gg \mathbf{role}\,\vec{S}.a(x).\mathbf{yield}\,\vec{S}.P'}$$

The typing rules of Table 2 are adapted accordingly. Of course, when dealing with output prefixes, we use $en_{\bar{u}}$ rather than en_a. Functions $en\_(\ldots)$ can be easily calculated by reducing the problem to a breath first search in a direct acyclic graph (cf. the full paper [3]).

The soundness of the modified judgment $\Gamma \vdash^{\mathcal{P}} A \gg A'$ now follows easily.

Proposition 4.1. *Let $(\mathcal{U};\mathcal{P})$ be a RBAC schema and Γ a typing environment respecting $\mathcal{U}$. Then, $\Gamma \vdash^{\mathcal{P}} A$ implies that $\Gamma \vdash^{\mathcal{P}} A \gg A$, while $\Gamma \vdash^{\mathcal{P}} A \gg A'$ implies that $\Gamma \vdash^{\mathcal{P}} A'$.*

[1] Several optimizations can of course be introduced to reduce the number of **role**/**yield**. For example, $\mathbf{role}\,R.a(x).\mathbf{yield}\,R.\mathbf{role}\,S.b(y).\mathbf{yield}\,S.P$ can be simplified in $\mathbf{role}\,R.a(x).b(x).\mathbf{yield}\,R.P$, whenever r^ole R enables inputs from both a and b. In general, such optimizations require complicated algorithms that we leave for future work.

To conclude, notice that there are other possible ways to find rôle sequences enabling inputs/outputs. For example, we can enforce the *least privilege* property. A system satisfies such a property if, whenever it performs an action, only the minimal set of permissions enabling the action are activated in the corresponding session. The approach presented above can be adapted to such requirements. The main change affects how functions $en\_(\ldots)$ are calculated, as the metrics to minimize is now $|\mathcal{P}(\vec{R})|$ rather than $|\vec{R}|$. Thus, the graph used to calculate these functions is weighted and records the number of permissions the activation of R adds to the current session's permissions. The best $\vec{R}$ is then extracted by using a minimal path algorithm.

Relocating Activities. We now investigate another application of our theory, viz. the transfer of a process from a user to another, which can be useful in order to minimize the number of users in a system. Balancing users' activity can also have a relevant economical impact: in a corporation, the management usually tries to raise productivity by optimizing and reassigning each employee's workload.

We now give an axiomatic way to infer judgments of the form

$$(\mathcal{U};\mathcal{P}) \triangleright r\{\!|P|\!\}_\rho \approx (\mathcal{U};\mathcal{P}) \triangleright s\{\!|P|\!\}_\rho.$$

This judgment says that the process P can be executed by r and s without affecting the overall system behaviour. Thus, the session $r\{\!|P|\!\}_\rho$ can be removed. If no other session of r is left in the system, then r itself can be removed. The procedure $\overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}}$ equates systems under the schema $(\mathcal{U};\mathcal{P})$. The rules defining it are given in Table 4, and generalize the equations given at the end of Section 3. We want to remark that a rule for relocating processes with restricted channels is missing. Indeed, the interplay between user names, restricted channel names and restricted channels is subtle. For example, consider the process $P \triangleq (\nu a:R)a^r\langle a^s\rangle$ and try to run it in users r, s and t. In the first case, no transition takes place; in the second case, a bound output takes place; in the third case, a free output takes place. Thus, relocating processes with restrictions breaks equivalences, in general.

As stated by the following Proposition, the procedure given above is a sound axiomatization for the judgment $(\mathcal{U};\mathcal{P}) \triangleright r\{\!|P|\!\}_\rho \approx (\mathcal{U};\mathcal{P}) \triangleright s\{\!|P|\!\}_\rho$.

Proposition 4.2. *If $r\{\!|P|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|P|\!\}_\rho$ then $(\mathcal{U};\mathcal{P}) \triangleright r\{\!|P|\!\}_\rho \approx (\mathcal{U};\mathcal{P}) \triangleright s\{\!|P|\!\}_\rho$.*

5 Related Work

To the best of our knowledge, no previous study building on process-calculi has ever been conducted on RBAC.

$$r\{\!|\,\mathbf{nil}\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,\mathbf{nil}\,|\!\}_\rho$$

$$\frac{r\{\!|\,P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_\rho \qquad r\{\!|\,Q\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,Q\,|\!\}_\rho}{r\{\!|\,P \mid Q\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P \mid Q\,|\!\}_\rho}$$

$$\frac{r\{\!|\,P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_\rho}{r\{\!|\,!P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,!P\,|\!\}_\rho}$$

$$\frac{r\{\!|\,P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_\rho}{r\{\!|\,[u = v]P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,[u = v]P\,|\!\}_\rho}$$

$$\frac{r\{\!|\,P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_\rho}{r\{\!|\,u\langle v\rangle.P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,u\langle v\rangle.P\,|\!\}_\rho}$$

$$\frac{\mathcal{U}(a^r) = \{R\} \qquad \mathcal{U}(a^s) = \{S\} \qquad \{R^?, S^?\} \cap \mathcal{P}(\rho) = \emptyset}{r\{\!|\,a(x).P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,a(x).P\,|\!\}_\rho}$$

$$\frac{R \notin \mathcal{U}(r) \cup \mathcal{U}(s)}{r\{\!|\,\mathbf{role}\,R.P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,\mathbf{role}\,R.P\,|\!\}_\rho}$$

$$\frac{R \in \mathcal{U}(r) \cap \mathcal{U}(s) \qquad r\{\!|\,P\,|\!\}_{\rho\cup\{R\}} \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_{\rho\cup\{R\}}}{r\{\!|\,\mathbf{role}\,R.P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,\mathbf{role}\,R.P\,|\!\}_\rho}$$

$$\frac{R \notin \rho}{r\{\!|\,\mathbf{yield}\,R.P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,\mathbf{yield}\,R.P\,|\!\}_\rho}$$

$$\frac{R \in \rho \qquad r\{\!|\,P\,|\!\}_{\rho-\{R\}} \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,P\,|\!\}_{\rho-\{R\}}}{r\{\!|\,\mathbf{yield}\,R.P\,|\!\}_\rho \overset{\mathcal{U}}{\underset{\mathcal{P}}{\equiv}} s\{\!|\,\mathbf{yield}\,R.P\,|\!\}_\rho}$$

Table 4. Relocating Activities

A number of papers have instead dealt with the formal specification and verification of RBAC schema. In [12, 19] formal methods are used only to verify the correctness of the schema definition but not of the whole system. In [19], the ALLOY language is used to detect possible conflicts in RBAC schemata supporting simultaneously delegation of authority and separation of duty. A constraint analyzer allows the schema validation to be computed automatically. In [12, 13], the authors use a graph transformation which combines an intuitive visual description of the RBAC schema with solid semantical foundations. Ahn et al. in [1] introduce a formal language for the specification of more sophisticated role-based authorization constraints, such as prohibition and obligation constraints. These approaches are complementary to ours: they can be integrated with our technique in order to verify the consistency of $(\mathcal{U}; \mathcal{P})$, but they do not give any hint about the correct execution of a system as our method does.

In [2], Bertino et al. develop a logical framework for reasoning about access control models. The framework is general enough to model discretionary, mandatory, and role-based access control models. Such a framework is useful for comparing the expressive power of the models, but it cannot be used to verify the correct execution of a system under a given schema.

Probably, the most related work, although not aiming at studying RBAC systems, is [4], insofar as rôles can be understood as (privilege) groups. *Groups* are introduced in *loc. cit.* as types for channels, and used to limit their visibility. A type system ensures that channels belonging to a fresh group can be only used by processes within the initial scope of the group. Thus, processes can access channels according to their physical distribution (with respect to group restrictions). In our work this feature is modified so that not only the place where the process runs (i.e., the user running the process) but also its execution history (i.e., the user session where the process runs) is relevant to execute an action. E.g., outputs over a^r of group R can be executed only by processes whose user r is such that $R^! \in \mathcal{P}(\mathcal{U}(r))$; moreover, such an action must be enabled by at least one of the rôles active in r's session. The set of such sessions changes according to the computation and, thus, the processes enabled to access a channel change dynamically. In this sense, this work can be seen as a calculus of *dynamic* groups.

Acknowledgments. This work has been partially supported by EU FET – Global Computing initiative, projects MIKADO IST-2001-32222 and MyThS IST-2001-32617, and by the FIRB project RBAU018RCZ_002. The funding bodies are not responsible for any use that might be made of the results presented here.

We also thank the CSFW anonymous referees for their valuable suggestions and comments.

References

[1] G.-J. Ahn and R. Sandhu. Role-based Authorization Constraints Specifi cation. *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] C. Braghin, D. Gorla, and V. Sassone. A distributed calculus for role-based access control. Full version of this paper, available as Research Report at `www.dsi.uniroma1.it/~gorla/publications.htm`.

[4] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *Proc. of CONCUR'00*, volume 1877 of *LNCS*, pages 365–379. Springer, 2000.

[5] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[6] D. Ferraiolo and D. Kuhn. Role-Based Access Control. In *Proc. of the NIST-NSA National Computer Security Conference*, pages 554–563, 1992.

[7] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[8] C. Fournet and C. Laneve. Bisimulations for the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, 2001.

[9] M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping. In *Proceedings of CATS '02*, volume 61 of *Electronic Notes on Theoretical Computer Science*. Elsevier Science Inc., New York, 2002. Full version to appear in *Mathematical Structures in Computer Science*.

[10] M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120, 2002.

[11] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[12] M. Koch, L. Mancini, and F. Parisi-Presicce. A Formal Model for Role-Based Access Control Using Graph Transformation. In *Proc. of 5th ESORICS*, volume 1895 of *LNCS*, pages 122–139. Springer, 2000.

[13] M. Koch, L. Mancini, and F. Parisi-Presicce. Decidability of Safety in Graph-based Models for Access Control. In *Proc. of 7th ESORICS*, volume 2502 of *LNCS*, pages 229–243. Springer, 2002.

[14] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proceedings of ICALP '98*, volume 1443 of *LNCS*, pages 856–867. Springer, 1998.

[15] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[16] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extract appeared in *Proceedings of LICS '93*: 376–385.

[17] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

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

[19] A. Schaad and J. Moffett. A Lightweight Approach to Specification and Analysis of Role-based Access Control Extensions. In *Proc. of 7th SACMAT*, pages 13–22. ACM Press, 2002.