# Discretionary Access Control with Code Migration

Chiara Polloni and Corrado Priami
Dipartimento Scientifico Tecnologico, Università di Verona
Ca' Vignal 2, Strada Le Grazie 1, I-37134 Verona, Italy – priami@sci.univr.it

## 1 Introduction

Mobile agents offer a new paradigm for distributed computation based on code migration. In this setting hosts need to identify each agent, to assign capabilities (migration, read, write, ...) to agents and to set agent resource consumption limits (CPU usage, memory and disk consumption, ...) to protect their resources from illegitimate usage. We describe in this work a discretionary access control method in which any host or any site decides the access rights to its resources.

This work is an attempt to formalize under which conditions migration of agents can occur and how agent access rights are assigned by destination hosts. The original contribution is the direct handling of multi-hop migration with respect to related works mentioned below.

A typing system for a distributed $\pi$-calculus is presented in [6] which guarantees that agents cannot access the resources of a system without first being granted the capability to do so. Types are assigned to processes and locations in [8]. The type of a process is determined by a set of capabilities and the type of a location is determined by the set of operations that processes can execute on that location. When a process migrate to a location, the location manager checks whether the operations that the process wants to execute are both in the set of its capabilities and in the set of operations of the location. The *spi*-calculus [1] is an extension of the $\pi$-calculus to formalise protocols for cryptography and authentication. Another distributed calculus to reason on security properties of mobile code is the Seal calculus [12]. The concept of ambient is introduced in [3] as a place where computations happen. Every ambient has a name and can be moved from a location to another. Processes have capabilities to enter or exit an ambient. A slight variant of mobile ambients to control interference via a type system is defined in [7]. Besides type systems, control flow analysis has been used as well to study security properties [2, 9]. A notion of causality is introduced in [11] to establish security properties of systems.

We stress that we are mainly concerned to *specify* the aspects to be dealt with when describing security policies of mobile systems. Hence, we leave some functions of our model unspecified and we consider them as parameters. For instance, the function $\sigma$ implementing the rules according to which resources are grant to reliable processes is a parameter of the hosts or of the locations (according to the granularity chosen). Different specifications of $\sigma$ results in different instances of the discretionary access control mechanism of the host.

## 2 The calculus

We extend the higher order $\pi$-calculus ($HO\pi$ for short) [10] by enriching names with an encoding of the addresses of their generators and by recording information on migration of processes during their execution. We first introduce the concept of *location*: a virtual site that resides on a physical site, thah contains a group of processes and that has associated a set of resources. We assume here that locations are uniquely named from a set $\mathcal{L}_{\mathcal{L}}$ interpreted as their addresses.

**Definition 2.1** *Let $\mathcal{N}$ be a countable infinite set of names ranged over by a, b, ... with $\tau \notin \mathcal{N}$. Let $\mathcal{M} = \mathcal{N} \cup \{a_l \mid a \in \mathcal{N}, l \in \mathcal{L_L}\}$ ranged over by $x, y, \ldots$. Let $\mathcal{V}$ be a set of process variables ranged over by $X$, $Y$, .... Let $K$ stand for a process or for a name, and let $U$ stand for a variable or for a name. Extended processes (denoted by $T, T'$, ... $\in \mathcal{R}$) are built according to the syntax*

$$P ::= \mathbf{0} \mid X \mid \pi.P \mid P + P \mid P|P \mid (\nu x)P \mid [x = y]P \mid A(U_1, ..., U_n)$$
$$\pi ::= x(U) \mid \overline{x}\langle L :: K\rangle \mid \tau$$
$$T ::= L :: T \mid T|T \mid (\nu x)T \mid [x = y]T \mid P$$

*where $x(U)$ is an input, $\overline{x}\langle L :: K\rangle$ an output ($x$ is the* subject *and $U, K$ are the* objects*), $\tau$ is a silent move, $A$ is an agent identifier, and $L$ is a list $\langle l_0, l_1, ..., l_n\rangle$ with elements in $\mathcal{L_L}$.*
*Given $\langle l_0, l_1, ..., l_n\rangle :: P$, with $P$ a sequential process, we interpret $l_0$ as the generator of $P$, and $l_1, ..., l_n$ as the sites on which $P$ migrated.*

Hereafter the trailing $\mathbf{0}$ will be omitted. Also, we assume a function $lc : \mathcal{R} \to \mathcal{L_L}$ that associates any process with its current location. We write $L_K, lc(T)$ $(L_K, L')$ to mean the list of location names $L_K$ extended to include $lc(T)$ $(L')$ in its rightmost position.

We let $L ::$ distribute over all the operators except for prefix which is handled by applying the operator $d_L$ to the process $\pi.P$ prefixed by $L ::$ (see the right equation in the third line below)

$$L :: (P + Q) = L :: P + L :: Q \qquad L :: (P|Q) = L :: P \mid L :: Q$$
$$L_0 :: L_1 :: P = L_1 :: P \qquad L :: (\nu x)P = (\nu x)L :: P$$
$$L :: [x = y]P = [x = y]L :: P \qquad L :: \pi.P = L :: d_L(\pi.P)$$

with

$$d_L(\mathbf{0}) = \mathbf{0} \qquad d_L(\tau.P) = \tau.d_L(P)$$
$$d_L(x(U).P) = x(U).d_L(P) \qquad d_L(\overline{x}K_l.P) = \overline{x}K_l.d_L(P)$$
$$d_L(\overline{x}\langle L' :: K\rangle.P) = \overline{x}\langle L', L :: K\rangle.d_L(P) \qquad d_L((\nu I)P) = (\nu I)d_L(P)$$
$$d_L(P + Q) = d_L(P) + d_L(Q) \qquad d_L(P|Q) = d_L(P)|d_L(Q)$$
$$d_L([x = y]P) = [x = y]d_L(P)$$

The first equation on the second line copes with process migration. For example, if the process placeholder $U$ in $L_0 :: x(U).U|L_1 :: \overline{x}\langle L_1 :: P\rangle$ is replaced by $L_1 :: P$ after the communication, the resulting process is $L_0 :: L_1 :: P|\mathbf{0}$. Then we are no more interested in the locations where the process that contains $U$ migrated, but we only need to know where $P$ went, i.e. $L_1$. So we delete $L_0$.

The last equation in the laws for $L ::$ deals with prefix. The operator $d_L$ registers the locations of $L$ in the objects of the output prefixes that are processes. It is needed for dealing with migration of inactive agents. Consider for instance the process $P = (\nu\, x, y)(l_0 :: \overline{x}\langle A\rangle|l_1 :: x(U).U|l_2 :: y(U).U)$ with $A = \overline{z}.\overline{y}\langle A_1\rangle$. Assume also that the locations of the three components from left to right are $l_0$, $l_1$ and $l_2$, respectively. After one communication we get $(\nu\, x, y)(\mathbf{0}|\{l_0, l_1\} :: A|l_2 :: y(U).U)$. Now $A$ performs $z$ and then activates $A_1$ via the next communication. Without the operator $d_L$, the system would register erroneously via $lc(\overline{y}\langle A_1\rangle)$ (see rules in Tab. 1) that $A_1$ has been created in location $l_1$. If location $l_2$ considers $l_1$ reliable, but $l_0$ not, the communication can occur. The operator $d_L$ prevents these drawbacks.

We enhance the labels of the standard transition system following [4, 5] with symbol $||_0$ $(||_1)$ that means that left (right) component of a parallel composition is moving. We label communication with a pair instead of a $\tau$ to record the components which interacted. We introduce also tag $+_0$ $(+_1)$ that records that the left (right) alternative of a nondeterministic choice is selected.

**Definition 2.2** *Let $\vartheta \in \{||_0, ||_1, +_0, +_1\}^*$. Then the set $\Theta$ of enhanced labels (with metavariable $\theta$) is defined by the following syntax*

$$\theta ::= \vartheta\mu \mid \vartheta\langle ||_0\ \vartheta_0\mu_0, ||_1\ \vartheta_1\mu_1\rangle$$

*with* $\mu_0 = x(U)$ *iff* $\mu_1 = \overline{x}\langle L :: K\rangle (resp. \overline{x}(\langle L :: K\rangle))$, *or vice versa. The function*

$$\ell(\vartheta\mu) = \mu \qquad\qquad \ell(\vartheta\langle \|_0\ \vartheta_0\mu_0, \|_1\ \vartheta_1\mu_1\rangle) = \tau$$

*maps enhanced labels to standard labels.*

We assume the usual functions $fn(\theta)$, $bn(\theta)$ and $n(\theta)$ that return the free, bound and all names of a label, respectively as well as the standard structural congruence on processes. We also assume the extensions of $fn$, $bn$ and $n$ to processes. For the late proved transition system for $HO\pi$ we refer the reader to [5].

Table 1 defines the operational semantics of extended processes, where we assume $fn(L :: P) = fn(P)$ and $(L :: P)\{x/y\} = L :: (P\{x/y\})$. Given a list $L$ of locations, we assume a function $first$ that extracts the first element of $L$ (i.e., the generator of the name or of the process prefixed).

When a name is sent as output of a communication, it is tagged with the current location (rule $Out_1$). No further tag is needed for a name created on another site (rule $Out_2$). Rule $Out_3$ updates a process created on the current location and output it including the tag for the current location in the first position of the list that will prefix $K$. Rule $Out_4$ copes with the output of a process created on a different site. Only a case for rules $Par$ and $Sum$ is reported. Also, we present $Open$, $Close$ and $Com$ rules only for two of their four schemata according to the output. The other cases should be obvious.

Note that rules $Close_2$ and $Com_2$ updates the sites on which processes migrated through function $lc$ by inserting the location of $T_1$ in the list of locations of $K$. Finally, we assume to start derivations of process $P$ in the transition system from $\epsilon :: P$, where $\epsilon$ denotes the empty list.

The names of locations are a mechanism of code signing used by the server to decide the safety of communications and access control. The signing is implemented by the runtime of the system (our semantics) and it is inaccessible to the application level (the syntax of the calculus is only modified by the compiler).

# 3 Interaction of processes

We here restrict the applicability of the rules for communications ($Com$ and $Close$) to forbid interactions when the sender is not a reliable process for the receiver. We assume that the location manager implements a function $\rho : \mathcal{L}_{\mathcal{L}} \to 2^{\mathcal{L}_{\mathcal{L}}}$ that provides all the locations judged reliable by a given $l \in \mathcal{L}_{\mathcal{L}}$. Relying on $\rho$, we define when a process is reliable.

**Definition 3.1** *A process* $\langle l_0, ..., l_n\rangle :: T \in \mathcal{R}$ *is* reliable for a location $l$ *if*

1. $l_0 \in \rho(l)$, *i.e.* $T$ *has been created on a reliable location; and*
2. $\forall i \in \{1, ..., n\} \,.\, l_i \in \rho(l)$, *i.e.* $T$ *is only migrated on reliable locations.*

*If only condition (1) holds, the process* $T$ *is* partially reliable, *otherwise it is not reliable.*

To establish which is the parallel component sent in a communication, we build abstract syntax trees of processes as binary trees whose nodes are $|$ or $+$ operators and whose leaves are sequential components of the whole process. Assume that their left (right) branches denote the left (right) component of parallel compositions or nondeterministic choices and label their arcs with tag $\|_0$ or $+_0$ (respectively $\|_1$ or $+_1$). Therefore any sub-process of a process is uniquely identified by a string $\vartheta \in \{+_0, +_1, \|_0, \|_1\}^*$. We define the selector operator $@\vartheta$ that applied to a process $T$ singles out its sub-process $T'$ at position $\vartheta$ in the abstract syntax tree.

**Definition 3.2** *Let* $\vartheta, \vartheta' \in \{\|_0, \|_1, +_0, +_1\}^*$, *and let* $\epsilon$ *the empty string. Then,* $T@\vartheta$ *is defined by*

$$T@\epsilon = T \quad L :: T@\vartheta = T@\vartheta \quad (\nu a)T@\vartheta = T@\vartheta \quad [x = y]T@\vartheta = T@\vartheta$$

$$(T|T')@\vartheta = \begin{cases} T@\vartheta' & if\ \vartheta = \|_0\ \vartheta' \\ T'@\vartheta' & if\ \vartheta = \|_1\ \vartheta' \end{cases} \qquad (T + T')@\vartheta = \begin{cases} T@\vartheta' & if\ \vartheta = +_0\vartheta' \\ T'@\vartheta' & if\ \vartheta = +_1\vartheta' \end{cases}$$

$$Act : L :: \mu.P \xrightarrow{\mu} L :: P,\ \mu \text{ not output} \qquad Res : \frac{T \xrightarrow{\theta}_I T'}{(\nu J)T \xrightarrow{\theta}_I (\nu J)T'}, J \cap n(\theta) = \emptyset$$

$$Out_1 : L :: \overline{x}K.P \xrightarrow{\overline{x}K_{lc(\overline{x}K.P)}} L :: P,\ K \in \mathcal{N}$$

$$Out_2 : L :: \overline{x}K_{l_K}.P \xrightarrow{\overline{x}K_{l_K}} L :: P,\ K \in \mathcal{M} - \mathcal{N}$$

$$Out_3 : L :: \overline{x}K.P \xrightarrow{\overline{x}\langle lc(\overline{x}K.P)::K\rangle} L :: P,\ K \in \mathcal{R}$$

$$Out_4 : L :: \overline{x}\langle l_0, \ldots, l_n :: K\rangle.P \xrightarrow{\overline{x}\langle l_0,\ldots,l_n::K\rangle} L :: P,\ K \in \mathcal{R}$$

$$Par_0 : \frac{T_0 \xrightarrow{\theta}_I T_0'}{T_0|T_1 \xrightarrow{\|_0\theta}_I T_0'|T_1}, (bn(\theta) \cup I) \cap fn(T_1) = \emptyset \quad Sum_0 : \frac{T_0 \xrightarrow{\theta}_I T_0'}{T_0 + T_1 \xrightarrow{+_0\theta}_I T_0'}$$

$$Open_1 : \frac{T \xrightarrow{\vartheta\overline{x}K_{l_K}} T'}{(\nu I)T \xrightarrow{\vartheta\overline{x}(K_{l_K})}_I T'}, x \notin I = \{K_{l_K}\}$$

$$Open_2 : \frac{T \xrightarrow{\vartheta\overline{x}\langle L_K::K\rangle} T'}{(\nu I)T \xrightarrow{\vartheta\overline{x}(L_K::K)}_I T'}, x \notin I \subseteq fn(K)$$

$$Close_1 : \frac{T_0 \xrightarrow{\vartheta\overline{x}K_{l_K}}_I T_0', T_1 \xrightarrow{\vartheta' x(U)} T_1'}{T_0|T_1 \xrightarrow{\langle\|_0\vartheta\overline{x}K_{l_K},\|_1\vartheta' x(K)\rangle} (\nu I)(T_0'|T_1'\{K_{l_k}/U\})}, K_{l_K} \notin fn(T_1), K \in \mathcal{M}$$

$$Close_2 : \frac{T_0 \xrightarrow{\vartheta\overline{x}\langle L_K::K\rangle}_I T_0', T_1 \xrightarrow{\vartheta' x(U)} T_1'}{T_0|T_1 \xrightarrow{\langle\|_0\vartheta\overline{x}\langle L_K::K\rangle,\|_1\vartheta' x(K)\rangle} (\nu I)(T_0'|T_1'\{(L_K, lc(T_1) :: K)/U\})} \begin{cases} fn(K) \cap fn(T_1) = \emptyset, \\ lc(T_1) \notin L_K \end{cases}$$

$$Com_1 : \frac{T_0 \xrightarrow{\vartheta\overline{x}K_{l_K}} T_0', T_1 \xrightarrow{\vartheta' x(U)} T_1'}{T_0|T_1 \xrightarrow{\langle\|_0\vartheta\overline{x}K_{l_K},\|_1\vartheta' x(U)\rangle} T_0'|T_1'\{K_{l_K}/U\}}, K \in \mathcal{M}$$

$$Com_2 : \frac{T_0 \xrightarrow{\vartheta\overline{x}\langle L_K::K\rangle} T_0', T_1 \xrightarrow{\vartheta' x(U)} T_1'}{T_0|T_1 \xrightarrow{\langle\|_0\vartheta\overline{x}\langle L_K::K\rangle,\|_1\vartheta' x(U)\rangle} T_0'|T_1'\{(L_K, lc(T_1) :: K)/U\}}, lc(T_1) \notin L_K$$

Table 1: Transition system for the extended $HO\pi$.

By abuse of notation we write $L_{T_0 @ \vartheta}$ for the list of locations that prefix the sub-process of $T_0$ at position $\vartheta$ within the abstract syntax tree of $T_0$.

Now a communication between the sender $T_0$ and the receiver $T_1$ is only allowed if $T_0$ is a reliable process for the location where $T_1$ resides. To implement this requirement we only need to add the side condition

$$(\forall l_i \in L_{T_0 @ \vartheta} \,.\, l_i \in \rho(lc(T_1)) \vee L_{T_0 @ \vartheta} = \epsilon \Rightarrow lc(T_0) \in \rho(lc(T_1))) \\ \wedge\, l_K \in \rho(lc(T_1)) \tag{1}$$

to rules $Com_1$ and $Close_1$, and the side condition

$$(\forall l_i \in L_{T_0 @ \vartheta} \,.\, l_i \in \rho(lc(T_1)) \vee L_{T_0 @ \vartheta} = \epsilon \Rightarrow lc(T_0) \in \rho(lc(T_1))) \\ \wedge\, first(L_K) \in \rho(lc(T_1)) \tag{2}$$

to rules $Com_2$ and $Close_2$. The side condition (1) says that the component of $T_0$ performing the output is judged reliable by the location hosting the component of $T_1$ performing the input, and that the name $K_{l_K}$ has been created on a reliable location. The second conjunct in (2) means that the process $K$ has been created on a reliable location. Besides processes we chose to trust names as well. This is because names in our setting can represent communication channels and accepting an untrusted name can give to a malicious agent potential access to the accepting site.

# 4 Resources access rights

When a process migrates on a location, it needs a set of access rights to the resources of the location that depends on the location where the process has been created and on the locations where migrated.

We introduce the numerable set $\mathcal{D}$ (with metavariable $r_i$) of resource identifiers uniquely numbered within a location and the set $\mathcal{T}$ (ranged over by $p_i$) of permissions. We then introduce three functions that we will use later to assign resources to processes. We left their precise definition as a parameter of our model. In fact we would fix a security politics by fixing them and we are looking for a theoretical template rather than for the formalization of specific realities. Of course, further work is needed to refine our template for security analysis. In particular, we need to instantiate our model to real systems (with actual definitions of our parameters $\sigma$, $\rho$, ... ) to test its applicability.

The function $\eta : \mathcal{R} \to 2^{\mathcal{D} \times 2^{\mathcal{T}}}$ describes the requests $\langle r, \{p_i\}\rangle$ of $T$. This function only depends on the process $T$ and thus it can be easily derived from the code of $T$.

The function $\sigma : \mathcal{L}_{\mathcal{L}} \times \mathcal{L}_{\mathcal{L}} \to 2^{\mathcal{D} \times 2^{\mathcal{T}}}$ describes the resources $\langle r, \{p_i\}\rangle$ that location $l$ grants to processes created or migrated on location $l'$. Function $\sigma$ encodes the decision strategy of the location manager of $l$ that in security literature is referred to as discretionary access control mechanism.

The function $\gamma : \mathcal{L}_{\mathcal{L}} \times \mathcal{R} \to 2^{\mathcal{D} \times 2^{\mathcal{T}}}$ returns the set of pairs $\langle r, \{p_i\}\rangle$ that location $l$ actually associates to process $T$ (to all sub-processes of $T$). The function $\gamma$ defines what the host gives to an agent after consulting the location manager ($\sigma$).

The set $safe_l \subseteq \mathcal{D} \times 2^{\mathcal{T}}$ is the largest set of pairs $\langle r, \{p_i\}\rangle$ that the location $l$ is willing to grant to partially reliable processes. We need $safe_l$ because the second conjunct of the side condition (2) only checks the generator of the agent $K$ that thus may result partially reliable.

For the sake of presentation, hereafter we write the first parameter of the functions $\sigma$ and $\gamma$ as a subscript of the function name yielding $\sigma_l$ and $\gamma_l$.

We now define when a process is safe. Intuitively, a process $T$ is safe for a location $l$ if the names of $T$ are created by $T$ itself or by reliable locations, the sub-processes in $T$ are reliable, $T$ is only migrated on reliable locations, and the resources requested by the sub-processes of $T$ are compatible with the ones that the location $l$ is willing to grant to $T$.

**Definition 4.1** *A process $L :: T$ is* safe for a location $l$ *if $T = \mathbf{0}$ or if the following conditions hold*

- $n(L :: T) = \{x_{l_x} \mid l_x = \epsilon \ \vee \ l_x = l \ \vee \ l_x \in \rho(l)\}$
- $p(L :: T) = \{(L' :: T') \subseteq T \mid (L' \subseteq \rho(l) \ \wedge \ \gamma_l(L' :: T') \subseteq \eta(T'))$
  $\vee \ (first(L') \in \rho(l) \ \wedge \ \gamma_l(T') \subseteq safe_l) \ \vee \ T' = \mathbf{0}\}$
- $L \subseteq \rho(l)$

*where $p(T) : \mathcal{R} \to 2^{\mathcal{R}}$ can be easily defined by structural induction to return the objects of the outputs that are processes and any sub-process of its argument.*

By abuse of notation we wrote $(L' :: T') \subseteq T$ in the above definition meaning that $(L' :: T')$ is a sub-process of $T$. The second condition $\gamma_l(L' :: T') \subseteq \eta(T')$ means that the location $l$ grants to $T$ (actually to its sub-process $T'$) the minimum set of resources that $T$ requires according to the Denning's principle of minimum resource consumption. Similarly, $\gamma_l(T') \subseteq safe_l$ states that a location $l$ grants to a partially reliable process at most the resources that $l$ considers of no danger.

Safety is a compositional property of processes.

**Proposition 4.2** *Let $T, T'$ be safe processes for a location $l$, and let $I$ be a set of names. Then $(\nu I)T$, $[x = y]T$, $T \mid T'$ and $T + T'$ are safe processes for $l$.*

Some comments are in order for compositionality of matching. It could be the case that $x$ and $y$ are located names, say $x_{l'}$ and $y_{l''}$. This can only happen if $x$ and $y$ have been received through a communication. Then the side condition (1) ensures that $l'$ and $l''$ are reliable for $lc(T)$.

We eventually prove that safety of processes is preserved under reduction steps.

**Theorem 4.3 (subject reduction)** *If $T$ is safe for a location $l$ and $T \xrightarrow{\theta} T'$, then $T'$ is safe as well.*

# References


[1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols - The Spi calculus. In *Proceedings of* $4^{th}$ *ACM Conference on Communications and Computer Security*. ACM press, 1997.

[2] C. Bodei, P. Degano, H.R. Nielson, and F. Nielson. Static analysis of processes for no read-up and no write-down. In *In Proceedings of 2nd FoSSaCS99, LNCS*, 1999.

[3] L. Cardelli and A. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98*, LNCS 1378, pages 140–155. Springer-Verlag, 1998.

[4] P. Degano and C. Priami. Proved trees. In *Proceedings of ICALP'92, LNCS 623*, pages 629–640. Springer-Verlag, 1992.

[5] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 216:237–270, 1999.

[6] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proceedings of POPL'98*, 1998.

[7] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of POPL'00*, 2000.

[8] R. De Nicola, G. Ferrari, R. Pugliese, and B. Venneri. Types for acess control. *Theoretical Computer Science*. To appear.

[9] H.R. Nielson, F. Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in mobile amients. In *Proceedings of CONCUR'99, LNCS 1664*, 1999.

[10] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[11] P. Sewell and J. Vitek. Secure composition of insecure components. In *Proceedings of CSFW-12*, 1999.

[12] J. Vitek and G. Castagna. Seal: a framework for secure mobile computations. In *Proceedings of Internet Programming Languages*. Springer-Verlag, 1999.