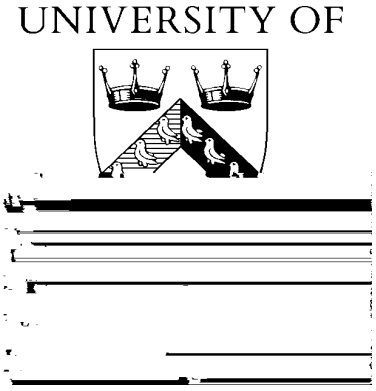
UNIVERSITY OF SUSSEX

COMPUTER SCIENCE



Security Policies as Membranes in

Security Policies as Membranes in Systems for Global Computing

Daniele Gorla, Matthew Hennessy and Vladimiro Sassone

Abstract. We propose a simple global computing framework, whose main concern is code migration. Systems are structured in sites, and each site is divided into two parts: a computing body, and a *membrane* which regulates the interactions between the computing body and the external environment. More precisely, membranes are filters which control access to the associated site, and they also rely on the well-established notion of *trust* between sites. We develop a basic theory to express and enforce security policies via membranes. Initially, these only control the actions incoming agents intend to perform locally. We then adapt the pre-adapt (e)-377t452ragnh esh-1.58266(o)-1.8744(o)-1.87515(n)-1.8858(b)-17572(p

1 Introduction

Computing is increasingly characterised by the global scale of applications and the ubiquity of interactions between mobile components. Among the main features of the forthcoming "global ubiquitous computing" paradigm we list *distribution* and *location awarness*, whereby code located at specific sites acts appropriately to local parameters and circumstances, that is, it is "context-aware"; *mobility*, whereby code is dispatched from site to site to increase flexibility and expressivity; *openness*, reflecting the nature of global networks and embodying the permeating hypothesis of localised, partial knowledge of the execution environment. Such systems present enormous difficulties, both technical and conceptual, and are currently more at the stage of exciting future prospectives than that of established of engineering practice. Two concerns, however, appear to clearly have a ever-reaching import: *security* and *mobility control*, arising respectively from openness and from massive code and resource migrations. They are the focus of the present paper.

We aim at classifying mobile components according to their behaviour, and at empowering sites with control capabilities which allow them to deny access to those agents whose behaviour does not conform to the site's *policy*. We see every site of a system

$$k[\![\ M \rhd P\ ]\!]$$

as an entity named k and structured in two layers: a *computing body* P, where programs run their code – possibly accessing local resources offered by the site – and a *membrane* M, which regulates the interactions between the computing body and the external environment. An agent P wishing to enter a site N must be verified by the membrane before it is given a chance to execute in N. If the preliminary check succeeds, the agent is allowed to execute, otherwise it is rejected. In other words, a membrane implements the policy each site wants to enforce locally, by ruling on the requests of access of the incoming agents. This can be easily expressed by a migration rule of the form:

$$k[\![\ M^k \rhd \mathbf{go}\, l.P \mid Q\ ]\!] \parallel l[\![\ M^l \rhd R\ ]\!] \to k[\![\ M^k \rhd Q\ ]\!] \parallel l[\![\ M^l \rhd P \mid R\ ]\!] \qquad \text{if } M^l \vdash^k P$$

The relevant parts here are P, the agent wishing to migrate from k to l, and l, the receiving site, which needs to be satisfied that P's behaviour complies with its policy. The latter is expressed by l's membrane, M^l. The judgement $M^l \vdash^k P$ represents l inspecting the incoming code to verify that it upholds M^l.

Observe that in the formulation above $M^l \vdash^k P$ represent a runtime check of all incoming agents. Because of our fundamental assumption of openendedness, such kind of checks, undesirable as they, cannot be avoided. In order to

tics as efficient as possible, we adopt a strategy which allows for efficient agent verification. Precisely, we adopt an elementary notion of *trust*, so that from the point of view of each l the set of sites is consistently partitioned between "good," "bad," and "unknown" sites. Then, in a situation like the one in the rule above, we assume that l will be willing to accept from a *trusted* site k a *k-certified digest* T of P's behaviour. We then modify the primitive **go** and the judgement $\vdash^k$ as in the refined migration rule below.

$$k[\![\, M^k \triangleright \mathbf{go}_{\mathsf{T}}\, l.P \mid Q \,]\!] \,\|\, l[\![\, M^l \triangleright R \,]\!] \to k[\![\, M^k \triangleright Q \,]\!] \;\|\; l[\![\, M^l \triangleright P \mid R \,]\!] \qquad \text{if } M^l \vdash^k_{\mathsf{T}} P$$

The notable difference is in $M^l \vdash^k_{\mathsf{T}} P$. Here, l verifies the entire code P against M^l *only if* it does not trust k, the signer of P's certificate T. Otherwise, it suffices for l to match M^l against the digest T carried by **go** together with P from k, so effectively shifting work from l to the originator of P.

Our main concern in this paper is to put the focus on the machinery a membrane should implement to enforce *different kinds* of policies. We first distill the simplest calculus which can conceivably convey our ideas and still support a non-trivial study. It is important to remark that we are abstracting from agents' local computations. These can be expressed in any of several well-known models for concurrency, for example CCS [13] or the π-calculus [14]. We are concerned,

where

- l is the site name
- P is the code currently running at l
- M is the membrane which implements the entry policy.

For convenience we assume that site names are unique in systems. Thus, in a given system we can identify the membrane associated with the site named l by M^l. We start with a very simple kind of policy, which we will then progressively enhance.

Definition 2.1 (Policies). A *policy* is any finite subset of **Act** $\cup$ **Loc**. For two policies T_1 and T_2, we write

$$\mathsf{T}_1 \ \texttt{enforces} \ \mathsf{T}_2$$

whenever $\mathsf{T}_1 \subseteq \mathsf{T}_2$. ■

Intuitively an agent conforms to a policy T at a given site if

- every action it performs at the site is contained in T
- it will only migrate to sites whose names are in T.

For example, conforming to the policy {`info`, `req`, **home**}, where `info`, `req` are actions and **home** a location, means that the only actions that will be performed are from the set {`info`, `req`} and migration will only occur, if at all, to the site **home**. With this interpretation of policies, our definition of the predicate `enforces` is also intuitive; if some code P conforms to the policy T_1 and T_1 `enforces` T_2 then P also automatically conforms to T_2.

The purpose of membranes is to enforce such policies on incoming agents. In other words, at a site $l[\![\, M \,\|\, Q \,]\!]$ wishing to enforce a policy T_{in}, the membrane M has to decide when to allow entry to an agent such as $\mathbf{go}_{\mathsf{T}} l.P$ from another site. There are two possibilities.

- The first is to syntactically check the code P against the policy T_{in}; an implementation would actually expect the agent to arrive with a proof of this fact, and this proof would be checked.
- The second would be to *trust* the agent that its code P conforms to the stated T and therefore only check that this conforms to the entry policy T_{in}. Assuming that checking one policy against another is more efficient than the code analysis, this would make entry formalities much easier.

Deciding on when to apply the second possibility presupposes a *trust management* framework for systems, which is the topic of much current research. To

simplify matters 35(i)0.677306(h)-1.8753.TJ-401.64-17.2801Td[(s)-4.130.6773-29(w)2.64796(

$$(\textbf{tc-empty})\quad \vdash \mathbf{nil} : \mathsf{T}$$

$$(\textbf{tc-act})\quad \frac{\vdash P : \mathsf{T}}{\vdash a.P : \mathsf{T}}\ a \in \mathsf{T}$$

$$(\textbf{tc-mig})\quad \frac{\vdash P : \mathsf{T}'}{\vdash \mathbf{go}_{\mathsf{T}'} l.P : \mathsf{T}}\ l \in \mathsf{T}$$

$$(\textbf{tc-repl})\quad \frac{\vdash P : \mathsf{T}}{\vdash !P : \mathsf{T}}$$

$$(\textbf{tc-par})\quad \frac{\vdash P : \mathsf{T} \qquad \vdash Q : \mathsf{T}}{\vdash P \mid Q : \mathsf{T}}$$

Figure 4. Typechecking incoming agents

The interesting reduction rule is the last one, (**r-mig**), governing migration; the agent $\mathbf{go}_{\mathsf{T}} l.P$ can migrate from site k to site l provided the predicate $M^l \vdash^k_{\mathsf{T}} P$ is true. This 'enabling' predicate formalises our discussion above on the role of the membrane M^l, and requires in turn a notion of code P satisfying a policy T,

$$\vdash P : \mathsf{T}$$

With such a notion, we can then define $M^l \vdash^k_{\mathsf{T}} P$ to be:

$$\textbf{if } M^l_t(k) = \mathsf{good} \textbf{ then } (\mathsf{T}\ \texttt{enforces}\ \mathsf{M}^l_{\mathsf{p}}) \textbf{ else } \vdash P : \mathsf{M}^l_{\mathsf{p}} \qquad (1)$$

In other words, if the target site l trusts the source site k, it trusts that the professed policy T is a faithful reflection of the behaviour of the incoming agent P, and then entry is gained provided that T enforces the entry policy $\mathsf{M}^l_{\mathsf{p}}$ (i.e., in this case, $\mathsf{T} \subseteq \mathsf{M}^l_{\mathsf{p}}$). Otherwise, if k can not be trusted, then the entire incoming code P has to be checked to ensure that it conforms to the entry policy, as expressed by the predicate $\vdash P : \mathsf{M}^l_{\mathsf{p}}$.

In Figure 4 we describe a simple inference system for c(i)1.9608(s)-4.13446(65(s)-4.135

cies, or more simply by checking the professed policy of trusted agents. The extent to which this strategy works depends, not surprisingly, on the quality of a site's trust management.

Example 2.1. Let **home** be a site name with the following trust function

$$M_t^h \ : \ \{\mathbf{alice}, \mathbf{bob}, \mathbf{secure}\} \quad \mapsto \quad \mathtt{good}\ .$$

Consider the system

$$S \triangleq \mathbf{home}[\![M^h \triangleright P^h]\!] \parallel \mathbf{bob}[\![M^b \triangleright P^b]\!] \parallel \mathbf{alice}[\![M^a \triangleright P^a]\!] \parallel \mathbf{secure}[\![M^s \triangleright P^s]\!]$$

in which the entry policy of **home**, M_p^h, is {`info`, `req`, **secure**}, and that of **secure**, M_p^s, is {`give`, **home**}. Since $M_t^h(\mathbf{bob}) = \mathtt{good}$, agents migrating from **bob** to **home** are trusted and only their digests are checked against the entry policy M_p^h. So, if P^b contains the agent

$$\mathbf{go}_{\mathsf{T}_1}\mathbf{home}.(\mathtt{take}.Q)$$

where T_1 `enforces` M_p^h, then the entry policy of **home** will be transgressed.

As another example, suppose **alice**, again trusted by **home**, contains the agent

$$\mathbf{go}_{\mathsf{T}_1}\mathbf{home}.(\mathtt{info}.\mathbf{go}_{\mathsf{T}_2}\mathbf{secure}.(\mathtt{take}.Q))$$

where T_2 is some policy which enforces the entry policy of **secure**, M_p^s. Again because T_1 `enforces` M_p^h, the migration is allowed from **alice** to **home**, and moreover the incoming agent conforms to the policy demanded of **home**. The second migration of the agent is also successful if **secure** trusts **home**: $M_t^s(\mathbf{home}) = \mathtt{good}$ and therefore only the digest T_2 is checked against the entry policy of **secure**. We then have the reduction

$$S \ \rightarrow^* \ \mathbf{home}[\![\ldots]\!] \parallel \mathbf{bob}[\![\ldots]\!] \parallel \mathbf{alice}[\![\ldots]\!] \parallel \mathbf{secure}[\![\ M^s \triangleright \mathtt{take}.Q \mid P^s\ ]\!]$$

in which now the entry policy of **secure** has been foiled. ■

The problem in this example is that the trust knowledge of **home** is faulty; it

(**wf-empty**)
$\vdash \mathbf{0} : \mathbf{ok}$

(**wf-g.site**)
$$\frac{\vdash P : \mathsf{M}_p}{\vdash l[\![M \rangle P]\!] : \mathbf{ok}}\ l \text{ trustworthy}$$

(**wf-par**)
$$\frac{\vdash N_1 : \mathbf{ok}, \qquad \vdash N_2 : \mathbf{ok}}{\vdash :}$$

the residual of those actions. The rules for the judgements

$$P \xrightarrow{\alpha} Q$$

where we let α to range over $\mathbf{Act} \cup \mathbf{Loc}$, are given in Figure 6, and are all straightforward. These judgements are then extended to

$$P \xrightarrow{\sigma} Q$$

where σ ranges over $(\mathbf{Act} \cup \mathbf{Loc})^*$, in the standard manner: $\sigma = \alpha_1, \ldots, \alpha_k$, when there exists P_0

$$\text{(tc-empty)}\quad \vdash \mathbf{nil} : \mathsf{T} \qquad \text{(tc-act)}\quad \frac{\vdash P : \mathsf{T}}{\vdash a.P : \mathsf{T} \cup \{\mathsf{a}\}} \qquad \text{(tc-mig)}\quad \frac{\vdash P : \mathsf{T}'}{\vdash \mathbf{go}_{\mathsf{T}'} l.P : \mathsf{T} \cup \{l\}}$$

$$\text{(tc-par)}\quad \frac{\vdash P : \mathsf{T}_1 \qquad \vdash Q : \mathsf{T}_2}{\vdash P \mid Q : \mathsf{T}_1 \cup \mathsf{T}_2} \qquad \text{(tc-repl)}\quad \frac{\vdash P : \mathsf{T}}{\vdash !P : \mathsf{T}'}\ \mathsf{T}^{\omega}\ \texttt{enforces}\ \mathsf{T}'$$

Figure 7. Typechecking with policies as Multisets

consider the system

$$S \triangleq \mathbf{mail\_serv}[\![M^{ms} \rangle P^{ms}$$

can be, in general, freely unfolded; hence, the actions they intend to locally perform can be iterated arbitrarily many times. For instance, agent

$$P \triangleq !\,\mathtt{send}$$

satisfies policy $\mathsf{T} \triangleq \{\mathtt{send}^{\omega}\}$. Notice that the new policy satisfaction judgement prevents the spamming virus of Example 3.1 from typechecking against the policy of **mail_serv** defined in Example 3.2.

The analysis of the previous section can also be repeated here but an appropriate notion of *well-formed* system is more difficult to formulate. The basic problem stems from the di

ϵ denotes the empty sequence of characters, α ranges over $\mathbf{Act} \cup \mathbf{Loc}$, '.' denotes concatenation, $\odot$ is the interleaving (or *shuffle*) operator and $^{\otimes}$ is its closure. Intuitively, if e represents the language L, then e

We want to conclude this section with two interesting properties enforceable by using automata.

Example 3.4. [Lock/Unlock] We have two actions, `lock` and `unlock`, with the constraint that each `lock` must be always followed by an `unlock`. Let $\Sigma_l = \Sigma - \{\texttt{lock}\}$ and $\Sigma_u = \Sigma - \{\texttt{unlock}\}$. Thus, the desired policy (written using a regular expression formalism) is

$$(\Sigma_l^*(\epsilon + \texttt{lock}.\Sigma_u^*.\texttt{unlock})^*$$

(**ti-empty**)
$\Vdash \mathbf{nil} : \emptyset$

(**ti-act**)
$$\frac{\Vdash P : \mathsf{T}}{\Vdash a.P : \mathsf{T} \cup \{a\}}$$

(**ti-mig**)
$$\frac{\Vdash P : \mathsf{T}'}{}$$

Therefore coherence, which is defined in terms of the trustworthiness of sites, is also preserved by reduction.

We outline the proof when the inference is deduced using rule (**r-mig**), a typical example. By hypothesis, $\vdash k[\![\, M^k \triangleright \mathbf{go}_{\mathsf{T}} l.P \mid Q \,]\!] : \mathbf{ok}$; this implies that $\vdash k[\![\, M^k \triangleright Q \,]\!] : \mathbf{ok}$. Thus, we only need to prove that $\vdash l[\![\, M^l \triangleright R \,]\!] : \mathbf{ok}$ and $M^l \vdash^k_{\mathsf{T}} P$ imply $\vdash l[\![\, M^l \triangleright P \mid R \,]\!] : \mathbf{ok}$. We have two possible situations:

l trustworthy. Judgment $\vdash R : \mathsf{M}^l_{\mathsf{p}}$ holds by hypothesis; judgment $\vdash P : \mathsf{M}^l_{\mathsf{p}}$ is implied by $M^l \vdash^k_{\mathsf{T}} P$. Indeed, because of the coherence hypothesis, $M^l_t(k) <: M^k_t(k)$. If $M^k_t(k) \neq \mathsf{good}$, then $M^l \vdash^k_{\mathsf{T}} P$ is exactly the required $\vdash P : \mathsf{M}^l_{\mathsf{p}}$. Otherwise, we know that $\vdash \mathbf{go}_{\mathsf{T}} l.P : \mathsf{M}^k_{\mathsf{p}}$; by rule (**tc-mig**) this implies that $\vdash P : \mathsf{T}$. Judgment $\vdash P : \mathsf{M}^l_{\mathsf{p}}$ is obtained by using Lemma A.1, since $M^l \vdash^k_{\mathsf{T}} P$ is defined to be T `enforces` $\mathsf{M}^l_{\mathsf{p}}$ (see (1) in Section 2.2). Thus, by using (**tc-par**), we obtain the desired $\vdash P|R : \mathsf{M}^l_{\mathsf{p}}$.

l not trustworthy. This case is simple, because rule (**wf-u.site**) always allows to derive $\vdash l[\![\, M^l \triangleright P \mid R \,]\!] : \mathbf{ok}$.

The case when (**r-act**) is used is similar, although simpler, and the case when rule (**r-par**) is used requires a simple inductive argument. Finally to prove the case when rule (**r-struct**) is used, we need to know that *coherency* of systems is preserved by structual equivalence; the proof of this fact, which is straightforward, is left to the reader. ■

Proof of Theorem 2.2 [Safety]: Let $l[\![\, M \triangleright P \,]\!]$ be a agent site in N such that $P \xrightarrow{\sigma} P'$. We have to prove that $\mathtt{act}(\sigma)$ `enforces` M_{p}. The statement is proved by induction over the length of σ. The base case, when $\sigma = \epsilon$, is trivial since $\mathtt{act}(\epsilon) = \emptyset$.

So we may assume $\sigma = \alpha\sigma'$ and $P \xrightarrow{\alpha} P'' \xrightarrow{\sigma'} P'$. Let us consider $P \xrightarrow{\alpha} P''$; by induction on $\xrightarrow{\alpha}$, we can prove that $\alpha \in \mathsf{M}_{\mathsf{p}}$ and that $\vdash l[\![\, M \triangleright P'' \,]\!] : \mathbf{ok}$. If the transition has been inferred by using rule (**lts-act**), then $P = a.P''$ and, by rule (**wf-g.site**), we have that $\vdash a.P'' : \mathsf{M}_{\mathsf{p}}$; by definition of rule (**tc-act**), we have the desired $a \in \mathsf{M}_{\mathsf{p}}$ and $\vdash P'' : \mathsf{M}_{\mathsf{p}}$. When (**lts-mig**) is used the argument is similar, and all other cases follow in a straightforward manner by induction.

Thus, we can now apply induction on the number of actions performed in $P'' \xrightarrow{\sigma'} P'$ and obtain that $\mathtt{act}(\sigma')$ `enforces` M_{p}. This sufficies to conclude that $\mathtt{act}(\sigma) = (\mathtt{act}(\sigma') \cup \{\alpha\})$ `enforces` M_{p}. ■

A.2 Proofs of Section 3.1

The proofs given in Appendix A.1 can be easily adapted to the setting in which entry policies are multisets. We outline only the main changes. First recall that the judgments $\vdash P : \mathsf{T}$ must be now inferred by using the rules in Figure 7 and

rule (**wf-g.site$_M$**) is used for well-formedness. Then, Lemma A.1 remains true in this revised setting.

Proof of Theorem 3.1 [Subject Reduction]: A straightforward adaptation of the corresponding proof in the previous section. The only significant change is to the case when a replication is unfolded via the rule (**r-struct**), i.e.

$$N \triangleq l[\![\, M \triangleright !P \mid Q \,]\!] \equiv l[\![\, M \triangleright P \mid !P \mid Q \,]\!] \rightarrow N'' \equiv N'$$

By hypothesis, $\vdash !P : \mathsf{M_p}$; therefore, by definition of rule (**tc-repl**), we have that $\vdash P : \mathsf{T}$ for some T such that T^ω `enforces` $\mathsf{M_p}$. Since T `enforces` T^ω and because of Lemma A.1, we have that $\vdash l[\![\, M \triangleright P \mid !P \mid Q \,]\!] :$ **ok**. By induction, $\vdash N'' :$ **ok**. It is easy to prove that this sufficies to obtain the desired $\vdash N' :$ **ok**. ■

Proof of Theorem 3.2 [Safety]: From the rule (**wf-g.site$_M$**) we know that $\vdash P_i : \mathsf{M_p}$, for all $i = 1, \ldots, n$. We now proceed by induction over $|\sigma|$. The base case is trivial. For the inductive case, we consider $\sigma = \alpha\sigma'$ and $P_i \xrightarrow{\alpha} P_i'' \xrightarrow{\sigma'}$

Proposition.

Proposition A.2.

1. A_1 `enforces` A_2 *can be calculated in polynomial time*
2. $\vdash P : \mathsf{A}$ *is decidable, but it is super-exponential*

Proof:

1. Let $\mathsf{A}_i = (S_i, \Sigma, s_0^i, F_i, \delta_i)$ and let $L_i = Acp(\mathsf{A}_i)$. By definition, we have to check whether $L_1 \subseteq L_2$ or not. This is equivalent to check whether $L_1 \cap \overline{L}_2 = \emptyset$. The following steps have been carried on by following [10].
   (a) calculate the automaton associated to $\overline{L}_2$. This can be done in $O(|S_2|)$ and the resulting automaton has $|S_2|$ states.
   (b) calculate the automaton associated to $L_1 \cap \overline{L}_2$. This can be done in $O(|S_1| \times |S_2| \times |\Sigma|)$ and creates an automaton A with $|S_1| \times |S_2|$ states.
   (c) Checking the emptyness of $L_1 \cap \overline{L}_2$ can be done by using a breath-first search that starts from the starting state of (the graph underlying) A and stops whenever a final state is reached. If no final state is reached, $L_1 \cap \overline{L}_2$ is empty. This can be done in $O(|S_1| \times |S_2| \times |\Sigma|)$.

   Thus, the overall complexity is $O(|S_1| \times |S_2| \times |\Sigma|)$.
2. It has been proved in [7] that each CRE e can be represented by a (labelled) Petri net, in that the language accepted by the Petri net is $lang(e)$. Now, we can easily construct a DFA accepting the complement of the language accepted by A (see item (a) of the previous proof). Now, we can construct the product between this DFA (that can be seen as a Petri net) and the Petri net associated to $CRE(P)$; this Petri net accepts $lang(CRE(P)) \cap \overline{Acp(\mathsf{A})}$ (see [17]). Now, the emptyness of this language can be solved with the algorithm for the reachability problem in corresponding Petri net. This problem has been proved decidable [12] and solvable in double-exponential time [3]. ■

We now prove the subject reduction theorem in the setting where types are DFAs. To this aim, we need to adapt Lemma A.1 and we need a very simple result on the languages associated to DFAs and processes.

Lemma A.3. *If* $\vdash P : \mathsf{A}$ *and* A `enforces` A'*, then* $\vdash P : \mathsf{A}'$*.*

Proof: By transitivity of subset inclusion. ■

Lemma A.4.

1. $\alpha\sigma \in Acp_s(\mathsf{A})$ *if and only if* $\sigma \in Acp_{\delta(s,\alpha}$ (e 1

Proof: Trivial. ■

Proof of Theorem 3.1 [Subject Reduction]: Now $\vdash N : \mathbf{ok}$ relies on rule ($\mathbf{wf\text{-}g.site_A}$). Again, the proof is by induction on the inference of $N \rightarrow N'$.

[5] U. Erlingsson and F. Schneider. SASI Enforcement of Security Policies: A Retrospective. In *Proc. of New Security Paradigms Workshop*, pages 87–95. ACM, 1999.

[6] C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In