Secured Information Flow for Asynchronous and Deterministic Objects

Isabelle Attali, Denis Caromel, Ludovic Henrio*, Felipe Luna Del Aguila

INRIA Sophia Antipolis, CNRS - I3S - Univ. Nice Sophia Antipolis

2004, Route des Lucioles, BP 93 - F-06902 Sophia Antipolis Cedex, France

{ia, caromel, henrio, fluna}@sophia.inria.fr

Abstract

We present in this article a precise security model for data confidentiality in the framework of asynchronous and deterministic objects. Our underlying programming model is based on active objects, asynchronous communications, and data-flow synchronizations. We extend its theoretical foundation, a calculus named ASP (Asynchronous Sequential Processes), with security levels attached to activities (active objects) and transmitted data.

We design a security model in order to guarantee the property of data confidentiality within an application; this security model takes advantages of both mandatory and discretionary access models. Then, we extend the formal semantics of ASP with predicate conditions that provides a formal security framework. At the same time, it makes it possible to dynamically check for unauthorized information flows. As a final result, we formally prove that, all authorized communication paths are secure, which means that no disclosure of information can happen. This theoretically-founded contribution may have a strong impact on distributed object-based applications, that are more and more present and confidentiality-demanding on the Internet.

*Currently at the University of Westminster - Harrow School of Computer Science - Harrow, HA1 3TP (UK).

1 Introduction

The main contribution of this work is to provide data confidentiality in communications and secure information flows by dynamic checks of asynchronous distributed object-based applications.

The proposed security model heavily relies on security policy rules with mandatory enforcements for the control of information flow. While information flows are generally verified statically [19, 3, 14, 13, 24, 21, 15, 9], our attention is focused on dynamic verifications. To achieve it, our model has an information control policy that includes discretionary rules, and because these rules are by nature dynamically enforceable, we can take advantage of the dynamic checks to carry out at the same time all mandatory checks. As another advantage of this approach, dynamic checks do not require to modify compilers, do not alter the programming languages, do not require modifications to existing source codes, and provide flexibility at run-time. Thus, dynamic checks fit well in a middleware layer which, in a non-intrusive manner, provides and ensures security services to upper-level applications.

Our underlying programming model [7] is based on active objects, asynchronous communications, and data-flow synchronizations. On the security side, security levels are used to independently tag the entities involved in the communication events: active objects and transmitted data. These "independent" tagging is however subject to discretionary rules. The combination of mandatory and discretionary rules allows to relax the strict controls imposed by the sole use of mandatory rules.

The advantages of our approach are twofold:

a sound foundation. This security model is founded on a strong theoretical background, the Asynchronous Sequential Processes (ASP) calculus [6], related to well-known formalisms [13, 12, 9, 8]. We extend the formal semantics of ASP with predicate conditions. This provides a formal basis to our model and, at the same time, makes it possible to dynamically check for unauthorized accesses. Finally, in order to prove the correctness of our security model, an intuitive secure information flow property is defined

and proved to be ensured by the application of access control model.

scalability and flexibility. We also target practical use of this model, with an implementation into middlewares, e.g. ProActive [20]. The granularity of our security model is defined in order to make it both efficient (because there is no security check inside an activity) and finely tunable: levels can be defined on activities because of the absence of shared memory but a specific level can be given for request parameters and created activities.

The potential impact of this work lies on the recent changes of paradigms in the area of distributed computing. The service oriented nature of ASP makes communications asymmetric (request and replies) and asynchronous (futures and wait-by-necessity). This security framework is, to our knowledge, the first to be adapted to the specificities of these communications.

Section 2 recalls our base model for objects and communications. Section 3 is the core of our contribution. Section 3.1 presents an access control model that can be implemented for ASP. The access control model conforms to both mandatory and discretionary controls, and applies to simple and independent actions (activity creation, request and reply transmissions). Section 3.2 defines and verifies an intuitive secure information flow property; from the previous independent actions, the concept of *information flow* is built, and also the notion of a *flow-path* is created by chaining basic flows; the property that an information flow is secure if all activity creations, requests and replies transmissions are secure concludes this section. Section 3.3 analyzes the specificity of our security model for service-oriented computing. Next, relation to existing work is discussed in section 4. Finally, section 5 concludes this article and draws future directions of research.

2 A Model for Objects and Communications

In this section, we introduce an object model and a model of communications; these two models are the foundation for our security model in section 3.

2.1 Object Model

The object model used in this work is based on the object-oriented paradigm, and because services are mostly intended to operate in an asynchronous mode, our work is based on the ASP (Asynchronous Sequential Processes) calculus [6]. The ASP calculus is an extension to the ς-calculus [1] where asynchronous communicating processes prevail. These processes, or *activities*, are running in parallel, but with their internal operations executed sequentially. What makes it outstanding are the concepts of *active objects*, *wait-by-necessity*, and *futures*. An *active object* is the extension of a "classical" object which acts like any other object but designed to be run remotely, in an activity, with his own sequential thread of execution. Table 1 presents the formal ASP language. The classical sequential reduction rules for the semantics description can be found in [6].

$a, b \in L$'	$::= x$	variable,
	$\mid [l_i = b_i; m_j = \varsigma(x_j, y_j)a_j]_{j\in 1..m}^{i\in 1..n}$	object,
	$\mid a.l_i$	field access,
	$\mid a.l_i := b$	field update,
	$\mid a.m_j(b)$	method call,
	$\mid clone(a)$	superficial copy,
	$\mid Active(a, m_j)$	object activation,
	$\mid Serve(m_1, .., m_n)^{n>0}$	service primitive,
	$\mid \iota$	location (not in source).
	$\mid a \Uparrow f, b$	a with continuation b

Table 1: The ASP calculus syntax

It is important to note that an activity is composed of only one active object, probably many passive objects and also probably many references to other active objects. As an example, Figure 1 shows activities α and β, where activity α has (among other entities) its own active object, two passive objects and a reference to activity β. Additionally, passive (or "classical") objects can only be referenced by objects belonging to the same activity, but any object can reference an active one.

One of the main contribution of ASP is the formalization of *futures* and request-reply patterns of communication. Futures are generalized references representing promises of reply that can be manipulated as a classical object (i.e. copied and transmitted inside and between activities) while their real value is not needed. An operation that needs the value of the object (e.g. a field access) is blocked until the necessary reply occurs.

This automatic and transparent synchronization mechanism is called *wait-by-necessity*.

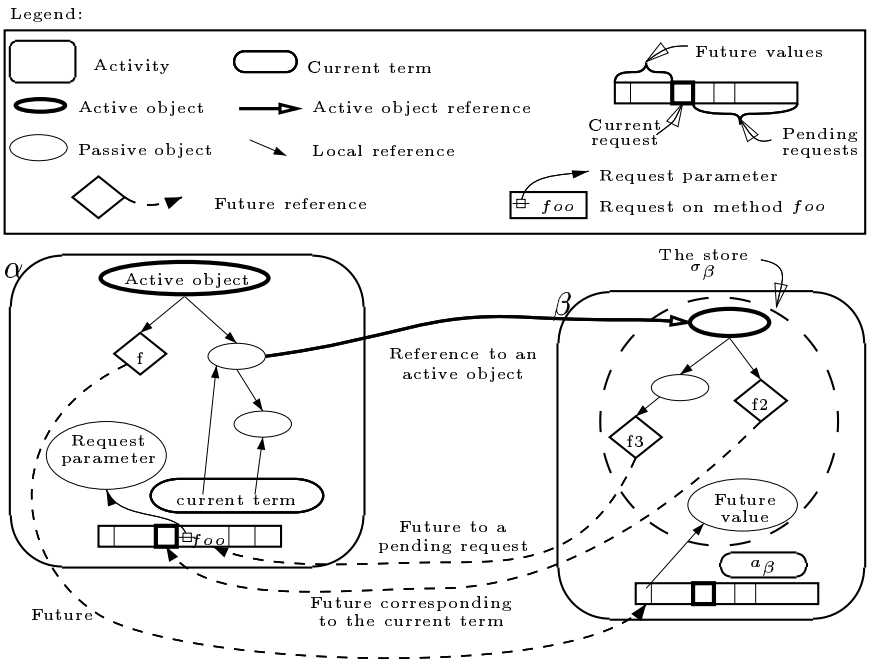

Figure 1: Example of a parallel configuration

Recalling the ASP syntax and semantics, a parallel configuration is a set of activities (α, β, $\gamma \in$ *Act*) of the form: P, Q ::= $\alpha[a; \sigma; \iota; F; R; f] \parallel \beta[\cdots] \parallel \cdots$

where *a* is the current term to be reduced; σ is the store containing all objects belonging to activity α; ι is the active object location; *F* is the list associating each result of a served request to its *futures*; *R* is the list of pending requests; and *f* is the future of the current term. Additional semantic notations are: locations ι and future identifiers f_i are local to an activity; a future $f_i^{\alpha \to \beta}$ is defined by an identifier f_i, a source activity α and a destination activity β according to the request; a reference to the active object of activity α is denoted by $AO(\alpha)$; and a reference to a future is denoted by $fut(f_i^{\alpha \to \beta})$.

2.2 Communication Model

The current ASP communication model is based on the parallel reduction rules shown in Table 2. These rules are based on a $copy(\iota, \sigma)$ operator which performs a deep copy of the store σ starting at location ι and

$$\frac{(a,\sigma) \rightarrow_S (a',\sigma') \qquad \rightarrow_S \text{ does not clone a future}}{\alpha[a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a';\sigma';\iota;F;R;f] \parallel P} \ (\text{LOCAL})$$

$$\frac{\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \qquad \sigma_\gamma = copy(\iota'',\sigma)}{\alpha[\mathcal{R}[Active(\iota'',m_j)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[\mathcal{R}[\iota'];\sigma';\iota;F;R;f] \parallel \gamma[\iota''.m_j();\sigma_\gamma;\iota'';\emptyset;\emptyset;\emptyset] \parallel P} \ (\text{NEWACT})$$

$$\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha\rightarrow\beta} \text{ new future} \qquad \iota_f \notin dom(\sigma_\alpha) \\ \sigma'_\beta = Copy\&Merge(\sigma_\alpha,\iota' \ ; \ \sigma_\beta,\iota'') \qquad \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha\rightarrow\beta})\} :: \sigma_\alpha\end{array}}{\begin{array}{l}\alpha[\mathcal{R}[\iota.m_j(\iota')];\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[\mathcal{R}[\iota_f];\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma'_\beta;\iota_\beta;F_\beta;R_\beta :: [m_j;\iota'';f_i^{\alpha\rightarrow\beta}];f_\beta] \parallel P\end{array}} \ (\text{REQUEST})$$

$$\frac{R = R' :: [m_j;\iota_r;f'] :: R'' \qquad m_j \in M \qquad \forall m \in M,\ m \notin R'}{\alpha[\mathcal{R}[Serve(M)];\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[\iota.m_j(\iota_r) \Uparrow f,\mathcal{R}[[]];\sigma;\iota;F;R' :: R'';f'] \parallel P} \ (\text{SERVE})$$

$$\frac{\iota' \notin dom(\sigma) \qquad F' = F :: \{f \mapsto \iota'\} \qquad \sigma' = Copy\&Merge(\sigma,\iota \ ; \ \sigma,\iota')}{\alpha[\iota \Uparrow f',a;\sigma;\iota;F;R;f] \parallel P \longrightarrow \alpha[a;\sigma';\iota;F';R;f'] \parallel P} \ (\text{ENDSERVICE})$$

$$\frac{\sigma_\alpha(\iota) = fut(f_i^{\gamma\rightarrow\beta}) \qquad F_\beta(f_i^{\gamma\rightarrow\beta}) = \iota_f \qquad \sigma'_\alpha = Copy\&Merge(\sigma_\beta,\iota_f \ ; \ \sigma_\alpha,\iota)}{\begin{array}{c}\alpha[a_\alpha;\sigma_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P \longrightarrow \\ \alpha[a_\alpha;\sigma'_\alpha;\iota_\alpha;F_\alpha;R_\alpha;f_\alpha] \parallel \beta[a_\beta;\sigma_\beta;\iota_\beta;F_\beta;R_\beta;f_\beta] \parallel P\end{array}} \ (\text{REPLY})$$

Table 2: Parallel reduction

$Copy\&Merge(\sigma_\beta,\iota' \ ; \ \sigma_\alpha,\iota)$ which appends, at the location ι of the store σ_α, a deep copy of the store σ_β starting at location ι'.

From these reduction rules, the only communication rules are *NEWACT*, *REQUEST* and *REPLY*. Only these rules are involved in the security framework. Hence, only these three rules are explained below. The *NEWACT* reduction rule creates a new activity γ containing the deep copy of an object, with empty values for the calculated future list, pending request list and current future identifier. A generalized reference to this activity $AO(\gamma)$ is stored in the source activity α. In the *REQUEST* reduction rule, activity α sends a new request to activity β. The new request $[m_j;\iota'';f_i^{\alpha\rightarrow\beta}]$ is made up of the target method m_j, the location ι'' of the argument passed in the request message, and the future identifier $f_i^{\alpha\rightarrow\beta}$ which will be related to the response resulting from the request. Note that in location ι'' there is a deep copy of the argument passed to the target method. The *REPLY* reduction rule, takes a reference to a future and updates it with its value. The reference to the future must exist in one activity α and the corresponding value must have been calculated

in another activity β. Note that a future $f_i^{\gamma \to \beta}$ can be updated in an activity different from the origin of the request ($\gamma \neq \alpha$).

3 The Security Model

In this section, we define a security model for our object model in order to guarantee the classic property of data confidentiality for multi-level security systems: a specific user with the appropriate clearance will be given access only to the information that he/she is allowed to handle. It is very important not to confuse this notion of data confidentiality with the (also called) confidentiality provided by encryption mechanisms (i.e. information obscuring).

We first recall usual notions of access control models and define our model in terms of entities and secured communications; which means that we formally extend ASP into Secure ASP. Then, we present our notion of secure information flow between activities with an important property for data confidentiality. We finally point out a fundamental aspect of the expressiveness of our model for service-oriented computing.

The security terminology used in access controls includes the subjects-objects relationship [23], and because the word "objects" can be confused with the object concept of Object-Oriented Programming, through the rest of this paper we will refer to this subject-object relationship as "subject-target".

3.1 Access Control Model

Access control models are generally classified into mandatory (MAC) or discretionary (DAC) models. The MAC model can be best described through the Multi-Level Security (MLS) model which is based on a lattice of security levels assigned to subjects and targets. Once levels are assigned, neither "normal users" nor processes can change them; making the system more secure against unauthorized access to the information. The MLS model is suited to address the confidentiality issue in information flows, but its inconvenient is that in certain cases it is less than adequate for practical systems. As for the case of DAC models, they are based on an access control matrix relating rights on subjects over targets, where the rights may be assigned at the

"discretion" of the "normal users" or their processes; this simple form of operation makes it more flexible compared to MLS.

The solution we propose is based on the concepts of MLS, with analogous notions of "no write-down" and "no read-up" taken from the model of Bell-LaPadula [4], but in order to avoid its total restrictiveness, it is modified and extended to cautiously include discretionary rules.

We begin by describing all entities involved in our security framework:

- $\mathcal{S}$ is the set of activities acting as subjects and/or targets, where $\alpha, \beta, \gamma, ... \in \mathcal{S}$; Bell-LaPadula's model is not applied as is, so subjects and targets are no longer classical persons/users and documents but activities (i.e., processes),
- $\mathcal{D}$ is the set of objects sent in the arguments of REQUESTS; a REQUEST is now written as $Rq_{\alpha\rightarrow\beta}(d)$ where $d = \sigma_\alpha(\iota') \in \mathcal{D}$,
- $\mathcal{R}$ is the set of objects associated to futures, and returned in REPLIES; a REPLY is now written as $Rp_{\beta\rightarrow\alpha}(r)$ where $r = \sigma_\beta(\iota_f) \in \mathcal{R}$.

Let $\mathcal{A}$ be the set of actions involved in the security mechanisms, i.e., REQUEST, REPLY and NEWACT. The following notations are added to ASP in order to take into account security aspects:

- Security levels λ are taken from a finite set $\mathcal{L}$, partially ordered by the relation $\leq, \forall i \in \mathcal{S} \cup \mathcal{D} \cup \mathcal{R},\ \lambda_i \in \mathcal{L}$,
- $\mathcal{T} = \mathcal{S} \times \mathcal{S} \times \mathcal{A}$ represents the authorized (access) transmissions,
- the matrix $\mathcal{M} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$ gives explicit (discretionary) rights to assign a level to a given data for a given action. For each subject-target pair, the matrix contain a set of authorized actions involving the assignation of a security level. $\mathcal{P}(\mathcal{A})$ classically denotes the set of sets of actions. The classic subject-target-action matrix is extended to include (allowed) security levels. This matrix can be implemented with a security policy file (e.g. XACML policy files [10]).

This results in the following characterization of actions (the modified semantics associated to the secured actions will be given in Table 4):

- $Nw(\gamma, \lambda_\gamma)$ is a modified activity creation rule (NEWACT) in order to assign a security level to an activity,
- $Rq_{\alpha\to\beta}(d, \lambda_{in})$ is a modified REQUEST transmission rule in order to allow for the tagging of the transmitted data with a security level (the programmer assigns a security level to data d),
- $Rp_{\beta\to\alpha}(r)$ is a REPLY transmission rule unchanged from original ASP.

To summarize, $a \in \mathcal{A}$ if and only if $a = Rq_{\alpha\to\beta}(d, \lambda_{in}) \vee a = Rp_{\beta\to\alpha}(r) \vee a = Nw(\gamma, \lambda_\gamma)$. Moreover, to be precise, $\mathcal{M}$ only has values in REQUEST or NEWACT. That is to say, it is not possible in our model to give to replies a discretionary right.

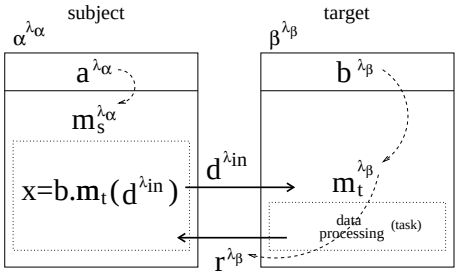

Figure 2: Communication between security-marked activities.

The security levels of subjects, targets, data and responses reflect the form of communications handled by activities. Figure 2 shows this form of communications. All activities are tagged with a security level and all objects and their methods therein contained will automatically inherit that level (a and b are objects, m_s is the calling method of the source activity, and m_t is the target method). Every data d used in a request transfer is also marked with a security level but this level is independent from that of the source activity.

It is the programmer responsibility to assign the security level to data d. Consequently, the level of the transmitted data will be added to the syntax of the method call (see Table 3). Even if it is not detailed in the following, a default behavior should consist in assigning the level of the sender activity to data d sent as

request parameter. In turn, every value r returned in a reply transfer will automatically be tagged with the security level of the target method (level inherited from the activity). This form of tagging allows output data to be independent of input data in the processing method, in other words, the security level for the output data does not depend on the level of the input data but on the processing of the data itself.

The conditions for secure communications in ASP are then derived and formalized according to our policy for communications:

Definition 1 (Secure activity creation)

$$\forall \alpha, \gamma \in \mathcal{S} : (\alpha, \gamma, Nw(\gamma, \lambda_\gamma)) \in \mathcal{T} \Longleftrightarrow (\lambda_\alpha \leq \lambda_\gamma) \vee Nw(\gamma, \lambda_\gamma) \in \mathcal{M}(\alpha, \gamma)$$

An activity creation $Nw(\gamma, \lambda_\gamma)$ is authorized if the activity is created with a level greater or equal to the one of the source activity. Else, if α wants to downgrade the data (i.e., the objects) used to create this new activity then there must be an explicit right allowing such an operation.

Definition 2 (Secure request transmission)

$$\forall \alpha, \beta \in \mathcal{S} : (\alpha, \beta, Rq_{\alpha \to \beta}(d, \lambda_{in})) \in \mathcal{T} \Longleftrightarrow (\lambda_{in} \leq \lambda_\beta) \wedge \left(\begin{array}{l} ((\lambda_\alpha > \lambda_{in}) \wedge Rq_{\alpha \to \beta}(d, \lambda_{in}) \in \mathcal{M}(\alpha, \beta)) \\ \vee \ (\lambda_\alpha \leq \lambda_{in}) \\ \vee \ \exists \gamma, \delta, f_i, \ d = fut(f_i^{\gamma \to \delta}) \end{array} \right)$$

The request transmission $Rq_{\alpha \to \beta}(d, \lambda_{in})$ is authorized to be emitted if the security level λ_{in} of the transmitted data d, is less than or equal to the security level λ_β of the target activity β; or, when source activity α with level λ_α tries to assign a level λ_{in} to data d (i.e. a data downgrading), there is an explicit right (discretionary rule $\mathcal{M}(\alpha, \beta)$) granting to the source activity α access to target activity β with this level.

The philosophy behind a secure request transmission is to "release" information only to a target which holds the appropriate clearance.

Further on, we also have a safe request transmission if the security level of data λ_{in} is greater or equal than that of the source λ_α. In that case, we have $\lambda_\alpha \leq \lambda_{in} \leq \lambda_\beta$, showing that activity α safely releases data d because d has a greater security level, and at the same time, activity β receives a lower level data.

Moreover, a safe request transmission is also achieved when handling future references $fut(f_i^{\gamma \to \delta})$ as data. Future references can be freely transmitted between activities because they do not hold any valuable information. We recall that values associated to futures hold information but future references only hold addresses or directions pointing to futures. In this sense, if a future reference is known, it does not mean we can directly get the future value, because anyway, the future value transmission will be performed by, and submitted to the security rules of, a *secure reply transmission*.

Definition 3 (Secure reply transmission)

$$\forall \alpha, \beta \in \mathcal{S} : (\alpha, \beta, Rp_{\beta \to \alpha}(r)) \in \mathcal{T} \iff (\lambda_\beta \leq \lambda_\alpha) \vee (\exists \gamma, \delta, f_i,\ r = fut(f_i^{\gamma \to \delta}))$$

The secure reply transmission REPLY $Rp_{\beta \to \alpha}(r)$ is authorized if the security level λ_β of target β is less than or equal than the security level λ_α of subject α, or if the transmitted result r only consists of a reference to a future $f_i^{\gamma \to \delta}$.

Table 3 shows the resulting secure ASP calculus. The secure ASP calculus syntax is based on the ASP syntax but security information is added to the activation and method call terms.

After attaching a security level to each activity, parallel confi gurations are now of the following form :

P, Q ::= $\alpha^{\lambda_\alpha}[a; \sigma; \iota; F; R; f] \parallel \beta^{\lambda_\beta}[\cdots] \parallel \cdots$

Finally, Table 4 presents the semantics of the secure parallel ASP calculus. These semantics rules ensure secure information fbw with secure requests and replies. They use the security information attached to the

$$
\begin{array}{lll}
a, b \in L' ::= x & & \text{variable,} \\
& | \, [l_i = b_i; m_j = \varsigma(x_j, y_j) a_j]_{j \in 1..m}^{i \in 1..n} & \text{object definition,} \\
& | \, a.l_i & \text{field access,} \\
& | \, a.l_i := b & \text{field update,} \\
& | \, a.m_j(b^{\lambda_{in}}) & \text{method call,} \\
& | \, clone(a) & \text{superficial copy,} \\
& | Active^{\lambda_a}(a, m_j) & \text{object activation,} \\
& | Serve(M) & \text{service primitive,} \\
& | \, \iota & \text{location} \\
& | a \Uparrow f, b & a \text{ with continuation } b
\end{array}
$$

Table 3: Secure ASP calculus

activation and method call terms (λ_a and λ_{in} in the $Nw(\gamma, \lambda_a)$ and $Rq_{\alpha \to \beta}(d, \lambda_{in})$ rules) to verify the secure transmission and activity creation defined before (Definitions 1, 2 and 3).

When a communication is not authorized, from the formal point of view, it is simply blocked. In practice a dedicated exception should be raised and appropriately handled.

$$
\frac{\begin{array}{c}\gamma \text{ fresh activity} \qquad \iota' \notin dom(\sigma) \qquad \sigma' = \{\iota' \mapsto AO(\gamma)\} :: \sigma \\ \sigma_\gamma = copy(\iota'', \sigma) \qquad (\alpha, \gamma, Nw(\gamma, \lambda_\gamma)) \in \mathcal{T}\end{array}}{\begin{array}{c}\alpha^\lambda[\mathcal{R}[Active^{\lambda_a}(\iota'', m_j)]; \sigma; \iota; F; R; f] \parallel P \longrightarrow \\ \alpha^\lambda[\mathcal{R}[\iota']; \sigma'; \iota; F; R; f] \parallel \gamma^{\lambda_a}[\iota''.m_j(); \sigma_\gamma; \iota''; \emptyset; \emptyset; \emptyset] \parallel P\end{array}} \text{(SecNEWACT)}
$$

$$
\frac{\begin{array}{c}\sigma_\alpha(\iota) = AO(\beta) \qquad \iota'' \notin dom(\sigma_\beta) \qquad f_i^{\alpha \to \beta} \text{ new future} \\ \iota_f \notin dom(\sigma_\alpha) \qquad \sigma'_\beta = Copy\&Merge(\sigma_\alpha, \iota' \, ; \, \sigma_\beta, \iota'') \\ \sigma'_\alpha = \{\iota_f \mapsto fut(f_i^{\alpha \to \beta})\} :: \sigma_\alpha \qquad (\alpha, \beta, Rq_{\alpha \to \beta}(\sigma_\alpha(\iota'), \lambda_{in})) \in \mathcal{T}\end{array}}{\begin{array}{l}\alpha^{\lambda_\alpha}[\mathcal{R}[\iota.m_j(\iota'^{\lambda_{in}})]; \sigma_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P \longrightarrow \\ \alpha^{\lambda_\alpha}[\mathcal{R}[\iota_f]; \sigma'_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma'_\beta; \iota_\beta; F_\beta; R_\beta :: [m_j; \iota''; f_i^{\alpha \to \beta}]; f_\beta] \parallel P\end{array}} \text{(SecREQUEST)}
$$

$$
\frac{\begin{array}{c}\sigma_\alpha(\iota) = fut(f_i^{\gamma \to \beta}) \qquad F_\beta(f_i^{\gamma \to \beta}) = \iota_f \\ \sigma'_\alpha = Copy\&Merge(\sigma_\beta, \iota_f \, ; \, \sigma_\alpha, \iota) \qquad (\beta, \alpha, Rp_{\beta \to \alpha}(\sigma_\beta(\iota_f))) \in \mathcal{T}\end{array}}{\begin{array}{c}\alpha^{\lambda_\alpha}[a_\alpha; \sigma_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P \longrightarrow \\ \alpha^{\lambda_\alpha}[a_\alpha; \sigma'_\alpha; \iota_\alpha; F_\alpha; R_\alpha; f_\alpha] \parallel \beta^{\lambda_\beta}[a_\beta; \sigma_\beta; \iota_\beta; F_\beta; R_\beta; f_\beta] \parallel P\end{array}} \text{(SecREPLY)}
$$

Table 4: Secure parallel reduction rules

3.2 Secure Information Flow in the Object Model

We formally define the notion of information flow between activities. The considered entities are activities together with their passive objects, and not passive objects on their own. Because activities can be distributed,

Non-Interference related notions [11] can not be directly applied to our model.

Next, system-wide information flows are described by a path. The path is the route along which the information travels, it is constructed by a chain of communicating activities where a subject activity is the starting-point and a target activity is the end-point of the path. Each information transmission observed on each activity will serve for the construction of a path. This path will be called *flow-path*.

Flow-paths fp are lists of activities ($fp := \alpha.\beta.\cdots$). They consist of the ordered list of transiting activities for a given information flow. For example $\varphi_{\gamma.\delta}(\alpha, \beta)$ means that some information has been transmitted from activity α to activity β through activities γ and δ. Concatenation of flow-paths fp and fp' is denoted by $fp.fp'$. By application of the security mechanisms to the non-secure information flow and flow-paths, a first property results: previous definition of information flow for an activity becomes secure if all activity creations, requests and replies transmissions are secure. Table 5 defines a *secure information flow*.

Definition 4 (Secure information flow) *An* elementary flow of information *is either based on the sending of a request, or on the sending of a reply, or on the creation of an activity. A* flow of information *is sequentially composed of several elementary flows. The flow-path of any flow of information is the concatenation of intermediate activities, it allows us to retrieve the original elementary flows. Secure information flow is built by concatenation of* elementary secure information flows *which are secured communications: secure* REQUEST, REPLY, *or* NEWACT.

$$\frac{(\alpha, \beta, Rq_{\alpha\to\beta}(\sigma(\iota'), \lambda_{in})) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\alpha, \beta)} \qquad \frac{(\beta, \alpha, Rp_{\beta\to\alpha}(\sigma_{\alpha}(\iota_f))) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\beta, \alpha)}$$

$$\frac{(\alpha, \gamma, Nw(\gamma, \lambda_{\gamma})) \in \mathcal{T}}{Sec\varphi_{\emptyset}(\alpha, \gamma)} \qquad \frac{Sec\varphi_{fp_1}(\alpha, \gamma) \qquad Sec\varphi_{fp_2}(\gamma, \beta)}{Sec\varphi_{fp_1.\gamma.fp_2}(\alpha, \beta)}$$

Table 5: Secure information flow

The following property states that a flow of information is secured if and only if it follows a secure path.

Property (Secure path for information flow) *A flow of information is secured if and only if it is composed of elementary secure information flows.*

$$Sec\varphi_{\gamma_1 \dots \gamma_n}(\alpha, \beta) \Longleftrightarrow Sec\varphi_{\emptyset}(\alpha, \gamma_1) \wedge Sec\varphi_{\emptyset}(\gamma_1, \gamma_2) \wedge \cdots \wedge Sec\varphi_{\emptyset}(\gamma_n, \beta)$$

The proof of this property is straightforwardly obtained by induction on the length of the information flow path and by a case analysis on the rules of Table 5.

This property does not take advantage of the MAC aspect of our model. Indeed, the same property could have been obtained with a purely DAC approach. This property rather shows that our specific and somehow less restrictive definition of information flow does not compromise secured information flow. A secured information flow property using the MAC aspect of our security policy is beyond the scope of this study. More generally, the study of the relation between mandatory and discretionary rules is closely related to the work of Bertino et al. [5].

Compared with other solutions, our secure information flow is the simple composition of a complex elementary flow. This results from the adaptation of the security formalism to a specific service-oriented framework. Complexity of the elementary flow comes from the asymmetric and asynchronous nature of ASP communications. Once such basic secure communications are ensured, the security of information flows is verified in a simple and intuitive manner. The soundness of secure information flow is thus ensured by a precise definition of *information* in the previous section and the fact that secure communications defined in Section 3.1 ensure that every information flow must verify the security policy.

3.3 Specificity of Service-Oriented Computing

Future references are first class objects and can be passed between activities (feature known as *automatic continuations*), thus they have an important consequence concerning the secured flows of information. Indeed, without automatic continuations, a flow of replies would directly follow the opposite path that a flow of requests, In other words, in a classical mandatory ruled system, a request-reply pattern of communications

can only occur between entities that have the same security level.

A first contribution of this paper is to authorize discretionary exceptions to these rules concerning level of data transmitted by requests. This allows some request-reply pattern to occur when the request sends non confidential data.

The possibility to transmit future references leads to a model well adapted specific to service-oriented computing. Let us focus on the configuration of Figure 3. Let us suppose that $\lambda_\delta \leq \lambda_\beta < \lambda_\gamma$ and consider futures f_2 and f_2'. If one did not have automatic continuation, γ could not return the value of future f_2 because it is a future reference to f_2'. Anyway, δ can reply to γ (because $\lambda_\delta < \lambda_\gamma$) but γ cannot forward this result value to β because $\lambda_\beta < \lambda_\gamma$. Indeed, transmitting the result from δ to β would require the following derivation to perform only authorized communications (**not authorized** means no reduction rule can be applied and thus the reply communication is considered as not secure – and then forbidden):

$$\dfrac{\dfrac{\lambda_\delta < \lambda_\gamma}{(\delta, \gamma, Rp_{\delta\to\gamma}(result)) \in \mathcal{T}} \quad \dfrac{\textbf{not authorized}}{(\gamma, \beta, Rp_{\gamma\to\beta}(r')) \notin \mathcal{T}}}{\dfrac{Sec\varphi_\emptyset(\delta,\gamma) \qquad Sec\varphi_\emptyset(\gamma,\beta)}{Sec\varphi_\gamma(\delta,\beta)}}$$

One could expect a better behavior because from a general point of view, the reply from δ to β should be authorized according to the security model. In the example of Figure 3 that means that β has sent a request and δ cannot reply only because there is an intermediate activity γ. Indeed if the request had not transited by γ the reply would be authorized.

The secured communication rules state that, as future references do not hold information, they can be freely transmitted; consequently γ can reply to β if the response is restricted to a future reference. Afterward, δ can reply directly to β because $\lambda_\delta \leq \lambda_\beta$, and β obtains the real value associated to f_2.

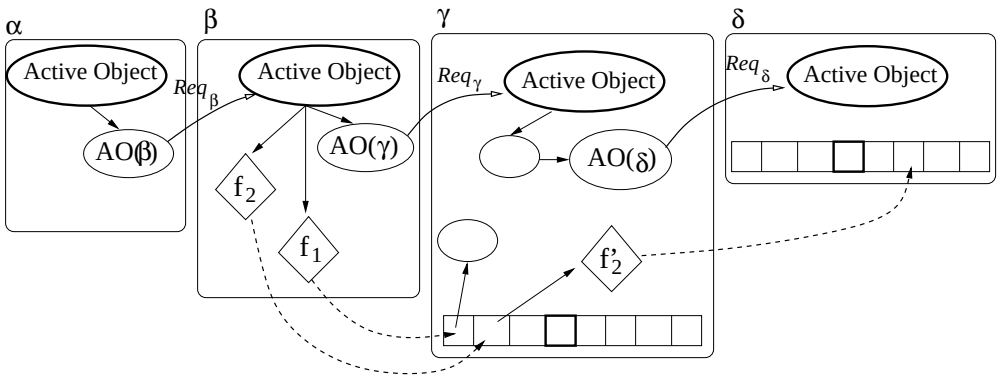

Figure 3: Information fbw between activities.

$$\frac{r = f_2'^{\gamma \to \delta}}{(\gamma, \beta, Rp_{\gamma \to \beta}(r)) \in \mathcal{T}} \qquad \frac{\lambda_\delta \leq \lambda_\beta}{(\delta, \beta, Rp_{\delta \to \beta}(r)) \in \mathcal{T}}$$

$$Sec\varphi_\emptyset(\gamma, \beta) \qquad Sec\varphi_\emptyset(\delta, \beta)$$

This example justifi es the possibility to freely transmit future references and demonstrates a communication pattern that would not be possible without the expressiveness of futures and the specifi c secured rules that exist in our mechanism.

Whereas, in ASP, the order in which future update occur has no consequence on the execution of a program, this example shows that in Secure ASP it is important to adopt a convenient future update strategy.

4 Related Work

Henessy and Riely present an extension of π-calculus [17, 18], a calculus aimed at distributed systems, extended through the use of (security) types [13]. We do not employ explicit channels to communicate but the read and write actions are analogous to receiving or sending requests and replies (a read when the request or reply is received, and a write when the request or reply is sent). Additionally, their processes may be analogous to our activities, but in general the security policies are not compatible nor they can be encoded in our model even with analogous notions.

Bertino et al. [5] treat exception-based information flow controls in object-oriented systems. They extend close work from Jajodia, Kogan, Sandhu [16] and Samarati et al. [22] to include operations (exceptions) normally not allowed by the strict security policy. They use an ACL (discretionary control) to operate on write and create actions, and with the permissive exceptions, they relax the strict policy imposed on those same actions. The use of exceptions to alter the strict applications mandatory rules of [5] is similar to the use of discretionary conditions in our framework. Both mechanisms allow one to bypass the rigorous application of strict/mandatory access controls.

Attali, Caromel and Contes present high-level rules which define a security policy for GRID applications built upon ProActive [2]. It is based on a discretionary approach where entities follow a hierarchical structure and relies on a Public Key Infrastructure. By comparison, our work focuses only on the communication actions, studies confidentiality in information flows specific to service-oriented applications, and uses both discretionary and mandatory approaches.

On the practical (implementation) side, Java-like languages that include information flow controls (as in [3, 19]) could be complemented with our model. They control information flows inside a program, so they could be enhanced to control all communication interactions with other local and non-local programs, in either distributed or cooperative systems.

5 Concluding Remarks and Future Work

We have presented a precise security model for the secure information flow in asynchronous and distributed object-based applications. The solution is mainly founded on three cornerstones: the concept of flow of information, security levels attached to activities, and the definition of security rules to be applied to all communications.

Flow of information was defined to take into account the way information can be handled in the asynchronous object model. This allowed us to demonstrate the specificity of our security mechanism: its application to service-oriented computing with replies by the means of futures.

The security policy (involving assignation, use, and definition of security levels) also takes into account the way information is handled in our model. When confidentiality is involved, there may exist high-level activities which may need to communicate with low-level activities (action that would normally be denied by the mandatory access rules of "no write down"). So by also tagging data with a security level we gain flexibility as the mandatory rules are not broken, and still, with the help of additional discretionary rules, we guarantee that this kind of actions are explicitly allowed. Communications are then controlled according to specific security rules. These rules are predefined in the case of mandatory rules, where the security levels of processes and data are always compared and applied; and in the case of discretionary rules, they are externally defined (for example in a file describing permissions for the whole system).

This security model has been implemented in the ProActive middleware [20] for distributed and mobile (Grid) computing, and is currently under evaluation on real-size examples for scalability and flexibility. Mobility of activities is part of the ASP model and the ProActive implementation, but has not been addressed in this article. In the future, it is also planned to use a role-based access control (RBAC) approach in order to extend and improve the discretionary access to activities.

References

[1] Martin Abadi and Luca Cardelli, *A theory of objects*, Springer-Verlag, 1996.

[2] Isabelle Attali, Denis Caromel, and Arnaud Contes, *Hierarchical and declarative security for grid applications*, International Conference On High Performance Computing, HIPC, Hyderabad, India, December 17-20 (Springer Verlag), Lecture Notes in Computer Science, LNCS, 2003.

[3] Anindya Banerjee and David A. Naumann, *Using access control for secure information flow in a java-like language*, 16th IEEE Computer Security Foundations Workshop (CSFW-16), July 2003.

[4] David E. Bell and Leonard J. LaPadula, *Secure computer system: Unified exposition and multics interpretation*, Technical Report MTR-2997 Rev. 1, The MITRE Corporation, Bedford, MA, March 1976.

[5] Elisa Bertino, Sabrina De Capitani di Vimercati, Elena Ferrari, and Pierangela Samarati, *Exception-based information flow control in object-oriented systems*, ACM Transactions on Information and System Security (TISSEC) **1** (1998), no. 1, 26–65.

[6] Denis Caromel, Ludovic Henrio, and Bernard Serpette, *Asynchronous and deterministic objects*, 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, ACM Press, 2004, pp. 123–134.

[7] Denis Caromel, Wilfried Klauser, and Julien Vayssiere, *Towards seamless computing and metacomputing in java*, Concurrency: Practice and Experience **10** (1998), no. 11-13, 1043–1061.

[8] Agostino Cortesi and Riccardo Focardi, *Information flow security in mobile ambients*, International Workshop on Concurrency and Coordination (ConCoord'01), Electronic Notes on Theoretical Computer Science, vol. 54, Elsevier, July 2001.

[9] Silvia Crafa, Michele Bugliesi, and Giuseppe Castagna, *Information flow security in boxed ambients*, Electronic Notes in Theoretical Computer Science, vol. 66:3, Elsevier, 2002.

[10] XACML eXtensible Access Control Markup Language, *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml*.

[11] Riccardo Focardi and Roberto Gorrieri, *Classification of security properties (part i: Information flow)*, Foundations of Security Analysis and Design (FOSAD 2000) - Tutorial Lectures, Lecture Notes in Computer Science, vol. 2171, Springer-Verlag, 2001, pp. 331–396.

[12] Matthew Hennessy, *The security picalculus and non-interference*, Journal of Logic and Algebraic Programming (2003), To Appear.

[13] Matthew Hennessy and James Riely, *Information flow vs. resource access in the asynchronous pi-calculus*, Computer Science Technical Report 2000:03, The University of Sussex, 2000.

[14] Peter Herrmann, *Information flow analysis of component-structured applications*, 17th Annual Computer Security Applications Conference, The Applied Computer Security Associates (ACSA), 2001.

[15] Kohei Honda, Vasco Vasconcelos, and Nobuko Yoshida, *Secure information flow as typed process behaviour*, Programming Languages and Systems, Lecture Notes in Computer Science, vol. 1782, Springer-Verlag, 2000.

[16] Sushil Jajodia, Boris Kogan, and Ravi S. Sandhu, *A multilevel secure object-oriented data model*, Information Security: An Integrated Collection of Essays (Marshall D. Abrams, Sushil Jajodia, and Harold J. Podell, eds.), IEEE Computer Society Press, 1995, pp. 596–616.

[17] Robin Milner, *Communicating and mobile systems: the π-calculus*, May 1999.

[18] Robin Milner, Joachim Parrow, and David Walker, *A calculus of mobile processes, part I/II*, **100** (1992), 1–77.

[19] Andrew C. Myers, *Jflow: Practical mostly-static information flow control*, 26th ACM Symposium on Principles of Programming Languages (POPL 99), ACM Press, January 1999, pp. 228–241.

[20] ProActive, *http://www-sop.inria.fr/oasis/proactive/*.

[21] Andrei Sabelfeld, *The impact of synchronisation on secure information flow in concurrent programs*, 4th International Conference on Perspectives of System Informatics, Lecture Notes in Computer Science, vol. 2244, Springer-Verlag, July 2001.

[22] Pierangela Samarati, Elisa Bertino, Alessandro Ciampichetti, and Sushil Jajodia, *Information flow control in object-oriented systems*, IEEE Transactions on Knowledge and Data Engineering **9** (1997), no. 4, 524–538.

[23] Pierangela Samarati and Sabrina De Capitani Di Vimercati, *Access control: Policies, models, and mechanisms*, Foundations of Security Analysis and Design : Tutorial Lectures, Lecture Notes in Computer Science, vol. 2171, Springer-Verlag, 2001, p. 137.

[24] Steve Zdancewic, Lantian Zheng, Nathaniel Nystrom, and Andrew C. Myers, *Untrusted hosts and confidentiality: Secure program partitioning*, 18th ACM Symposium on Operating System Principles (SOSP '01), vol. 35, ACM Press, October 2001, pp. 1–14.