8

Towards a mobile LOTOS

E. Najm, J-B. Stefani**, A. Février**

(*) ENST
Ecole Nationale Supérieure des Télécommunications
46, rue Barrault
75013 Paris
FRANCE
{najm|fevrier}@res.enst.fr

(**) CNET
Centre National d'Etudes des Télécommunications
38-40, rue du Général Leclerc
92131 Issy-Les-Moulineaux
FRANCE
stefani@issy.cnet.fr

Abstract

LOTOS is known not to address elegantly some of the modelling problems raised by modern distributed systems. Mobility and dynamic reconfiguration are among the essential features of these systems that are not properly addressed by this language. The present document is an attempt to solve this problem. We present M-LOTOS a mobile enhancement of LOTOS which preserves its other specification styles. M-LOTOS draws upon research in the area of mobile process algebras in general and upon Milner's & all π-calculus in particular.

Keywords

Formal semantics, formal description techniques, FDT, Open Distributed Processing, pi-calculus, bisimulation, Computational Model, LOTOS, dynamic reconfiguration, gate passing

1. INTRODUCTION AND MOTIVATION

Modern distributed systems are characterised by their flexibility, mobility and adaptability. They should potentially offer a flexible environment for modification and extension. Components should be included and removed to attend to or cope with changes in the system, the environment or the user's requirements. For instance, a system should be designed to dynamically and transparently replace a faulty element. It should allow also for new services to be introduced, made public, traded and used. Performance issues, for instance in fast communication protocols, require that software components are dynamically combined and put to work: depending on the context, only the required protocol functions should be activated. Migration of components is also an important feature which enhances the performance and the availability of systems.

G. V. Bochmann et al. (eds.), *Formal Description Techniques VIII*

Since July 94, the standardisation of Formal Description Techniques (FDTs) within ISO has become part of the Open Distributed Processing (ODP) standardisation committee (SC21/ WG7). ODP aims at defining a generic framework and architecture for the development of open distributed systems. Thus, supporting the formal design of open distributed systems is a new challenge for FDTs. Indeed, the ISO-FDT group of experts is now working on the standardisation of an extension of LOTOS - temporarly called E-LOTOS, which is targeted, among other things, at providing support to the design of ODP systems. This group has established a list of desirable features together with a list of requirements that E-LOTOS should aim to fulfil. Aspects related to real-time, constructive data representations and modularity are being actively studied. Dynamic reconfiguration of communication structures (e.g., gate passing between *processes*) is now receiving more attention.

In the ODP framework, which draws heavily on recent distributed system research, the basic concepts are object and object composition. As defined in this framework, an object is a model of a self-contained, well identified, part of a system, which encapsulates data and behaviour, and can interact with other parts of the system through a set of interfaces. A configuration (or composition) is a collection of objects which can interact among themselves and with their environment. This notion of configuration is a very general one: objects in a configuration can be created, destroyed, or can learn about other objects, can be bound with other objects, and can change their potential interaction partners. For the time being, there is no adequate formal model of such a notion that could capture all the required dynamicity in the idea of configuration. There are, however, both specification and programming languages which do exhibit some form of dynamic reconfiguration. One type of such languages is given by Milner's π-calculus, or calculus of mobile *processes*. The π-calculus extends CCS by allowing the passing of gate names as interaction parameters.

The aim of the present paper is to introduce M-LOTOS a mobile extension of LOTOS based on the π-calculus. M-LOTOS is an upward compatible extension of LOTOS which is orthogonal to other extensions currently discussed in the E-LOTOS group, like e.g. real time. The paper is structured as follows, in section 2 we give the rationale for the design choices, in sections 3 and 4 we present the syntax and an informal introduction to the semantics of M-LOTOS. Section 5 deals with the formal semantics. Section 6 is devoted to an example and in section 7 we conclude.

2. INTRODUCING MOBILITY IN LOTOS

The mere extension by which gates are allowed to be passed like values between LOTOS *processes* is not sufficient to express the intended dynamic reconfiguration behaviour. This is because the communication capacities of a LOTOS *processes* do not depend only on the gates it possesses, but also on the gate parameters of the parallel operators applied to it. Thus a workable extension is one which also provides for the CSS-like (and π-calculus) parallel composition. Knowing that the CCS parallel and restriction operators go hand in hand, one gets to the conclusion that a natural solution is to combine the π-calculus and LOTOS in one coherent notation. Thus our equation: M-LOTOS = π-calculus + LOTOS. In fact, the combination of these two calculi is achieved using a new operator called *embedding*, denoted <_>, which turns a LOTOS *behaviour-expression*, say *B*, into the M-LOTOS expression <*B*>. Expressions of the form <*B*> are called *agent-expressions*. *Agent-expressions* can be combined only using the two π-calculus operators: parallel composition and restriction. For instance, if B_1

and B_2 are 2 LOTOS *behaviour-expressions*, then $<B_1> \mid <B_2>$ and **restrict** g **to** $(<B_1> \mid <B_2>)$ are three M-LOTOS *agent-expressions*. $<B>$ is the simplest form of an *agent-expression*. The function of the embedding operator, <_>, is to put a boundary around a LOTOS *process*, thus making it behave externally like a π-calculus *agent*. An action of $<B>$ may result from a multi-synchronisation of *sub-processes* of B but can only perform a binary rendez-vous with its environment. It is interesting to note that the embedding operator brings a new style of specification in LOTOS. Indeed, one may in M-LOTOS distinguish between two levels: the *local* level where *processes* may be tightly coupled using a constraint oriented style, and the *distributed* level where *agents* are loosely coupled and interact by pairs.

3. SYNTAX OF M-LOTOS

The syntax of M-LOTOS is given below by the production rules (for the sake of clarity, but without any loss of generality, we base our discussion on an essential subset of LOTOS):

C ::= $< B >$
| **restrict** $g_1, ..., g_n$ **to** C
| $C_1 \mid C_2$
| $\hat{A}\,[h_1, ..., h_k](E_1, ..., E_m)$

c ::= **i** | **i new** C
| a | a **new** C
| a [*pred*] | a **new** C [*pred*]

a ::= $g\ off_1 ... off_n$

off ::= $!E$ | $?x{:}t$ | $!g$ | $?h{:}gid$

B ::= **stop**
| c ; B
| B_1 [] B_2
| B_1 |[$g_1, ..., g_n$]| B_2
| **exit**
| $B_1 >> B_2$
| B_1 [> B_2
| **hide** $g_1, ..., g_n$ **in** B
| $A\,[h_1, ..., h_k](E_1, ..., E_m)$

Some remarks are in order:

(i) C, C_1 and C_2 are generic M-LOTOS *agent-expressions*.

(ii) $\hat{A}\,[h_1, ..., h_k](E_1, ..., E_m)$ is an instantiation of an *agent* definition, thus, there exists a unique definition of the form: **agent** $\hat{A}\,[h_1, ..., h_k](x_1{:}t_1, ..., x_n{:}t_n) = C$ **endagent,**

(iii) B, B_1 and B_2 are generic *behaviour-expressions* of the LOTOS sub-language of M-LOTOS, called LOTOS^M in the sequel. LOTOS^M *behaviour-expressions* are similar to LOTOS *behaviour-expressions* except for actions: LOTOS^M actions use gate passing in offers and may contain creations of new *agents*. Creation of a new *agent* is written: a **new** C. The intuitive meaning of action a **new** C is: perform action a and, as a side effect, create *agent* C. Examples of this construct will be given later.

(iv) the value parameter list of *process* or *agent* definitions may contain gates which are then declared with the predefined type *gid*. A gate cannot be declared in both the gate parameter list and the value parameter list. If a gate is declared in a value parameter list of a *agent* of *process* definition then there is a restriction in the way it may be used in an action. For instance, if h is such a gate and $c = g\ off_1 ... off_n$ [*pred*] **new** C an action contained in

the body of one such definition, then *h* can occur in offers off_i, in the predicate *pred* or in the *agent-expression C*, but *h* cannot be the interaction gate: $h \neq g$. Note that there is no restriction on the use of *h* within the *agent C* and thus it can be an interaction gate in a spawned *agent*.

(v) hide and restriction are creators of new (and private) names. These two operators are parametrised with a gate list. As usual, when applied to an expression, these two operators, bound all the free gates of that expression that appear in their gate parameter list.

4. INFORMAL INTRODUCTION TO THE SEMANTICS OF M-LOTOS

Following a now classical approach, the operational meaning of an M-LOTOS *agent-expression C* is its set of possible derivations deriv(*C*) where a derivation of deriv(*C*) is a couple, (α, *C'*), made of an action α and a *agent-expression C'*. The statement (α, *C'*) $\in$ deriv(*C*) means: *C* is able to perform action α and, doing so, it evolves into *agent-expression C'*. $C \xrightarrow{\alpha} C'$ is an equivalent notation for the statement: (α, *C'*) $\in$ deriv(*C*). In these derivations, actions are computed using an *early binding* of gate names, a method which has been used in one of the many formulations of the π-calculus semantics [Milner91] (please note that familiarity with the π-calculus is not a prerequisite for what follows).

The semantics of *agent-expressions* is evaluated in two steps: first at the LOTOS^M level then at the M-LOTOS level. At the LOTOS^M level we compute derivations of the form: $B \xrightarrow[C]{\alpha} B'$, where *B* and *B'* are LOTOS^M expressions, α is an action and *C* is a *agent-expression*. Derivation $B \xrightarrow[C]{\alpha} B'$ means: *B* performs action α, spawns *agent C*, then evolves to *B'*. The LOTOS^M derivations for *B* and the M-LOTOS derivations for *agent* <*B* > are related by an inference rule:

$$\frac{B \xrightarrow[C]{\alpha} B'}{< B > \xrightarrow{\alpha} \textbf{restrict } g_1, ..., g_n \textbf{ to } (< B' > | C)}$$

where the gates g_1, ..., g_n are private communication links between < *B'* > and *C* (it will be shown later how gates g_1, ..., g_n are computed). The set of actions is given by the production rules:

$$\alpha ::= i \mid \delta \mid g \mid g\ \omega_1 ... \omega_n$$
$$\omega ::= v \mid !h \mid ?h \mid !\underline{h} \mid ?\underline{h}$$

where, δ is the successful termination gate (which cannot be passed between *processes*), g, h are gate names, and ω, ω_1, ..., ω_n *observed* offers. An *observed* offer is either a value v (which is not a gate) or a *gate* offer which can be one of the following four possible forms: !h, ?h , !$\underline{h}$, ?$\underline{h}$. These forms are explained hereafter:

(i) ! h : offering a *public* gate name h, i.e., gate h occurs free in the offering expression. When we write $P \xrightarrow{g!h} P'$ we can conclude that h occurs free in *P*. Example: g!h ; $B \xrightarrow{g!h} B$.

(ii) ! $\underline{h}$: offering a *fresh* gate name h. The offered gate is a new gate just being exported. When we write $P \xrightarrow{g!\underline{h}} P'$ we can conclude that h is bound in *P* by a hide operator. Example: take P = **hide** h **in** g!h; *B* then we may write the derivation: $P \xrightarrow{g!\underline{h}} B$. In this example h is a place holder: any other name obtained by α-converting *P* would do. Thus we also can write the following derivation: $P \xrightarrow{g!\underline{u}} B\{u/h\}$ provided name u is not free in *P*.

(iii) ? h : receiving a *fresh* gate name. In this case h is a place holder for the received gate. With this type of offer we are prepared to receive a gate that has just has been created by another *process*. In a derivation: $P \xrightarrow{g\,?\,h} P'$, we have $h \notin fg(P)$ (h is not a free gate of P). Example: g ?x:gid ; $B \xrightarrow{g\,?\,h} B\{h/x\}$ where h is any name with $h \notin fg(g\,?\,x{:}gid\,;B)$. Note that one possible derivation from g ? x:gid ; B is simply: g ? x:gid ; $B \xrightarrow{g\,?\,x} B$.

(iv) ? h : receiving any gate name. This form is to be compared with the previous one. These two forms are possible from the action of reception of a gate (such as g ? x:gid). Such an action implies that a *process* is ready to receive any name from the sender (?h), or a name just created by the sender (?h). Example, take P = g ? x:gid ; B then $P \xrightarrow{g?\,h} B\{h/x\}$ where h may occur free in B.

Hereafter we give a collection of simple derivations. Then we will turn to presenting illustrations of *agent* creation, action synchronisations and parallel composition of *agents*.

Example 1

a ? x:Nat ; b ! x ; **stop** —a 0→ b ! 0 ; **stop** : this derivation is obtained from the usual semantics of LOTOS. It describes the acceptance of the natural number 0 on gate a (this is not the only possible derivation as any natural number can be accepted on gate a).

Example 2

< a ? x:Nat ; b ! x ; **stop** > —a 0→ < b ! 0 ; **stop** > : the same as above, but this time we are considering an *agent-expression*.

Example 3

< g ? y:gid ; g' ! h ; **stop** > —g ? h→ < g' ! h ; **stop** > : accepting a public gate h on gate g. Note that, in this derivation, the accepted gate is already being used in the expression (it is offered on gate g'). This is not the only possible derivation from action g ? y:gid (see example 4 below).

Example 4

< g ? y:gid ; g' ! h ; **stop** > —g ? h'→ < g' ! h ; **stop** > : accepting a *new* gate h' on gate g. Since h is underscored we have h' ≠ h (this property is enforced by the semantics). Note that the prefix: g ? y:gid generates two kinds of actions: either the acceptance of any gate (this was illustrated in example 3), or the acceptance of a *fresh* gate which could be any name which is not free in the accepting expression.

Example 5

< **hide** y **in** g ! y ; g' ! y ; **stop** > —g ! h→ < g' ! h ; **stop** > : offering a new gate on gate g. The name h chosen for this new gate is such that it is not already used free in the sending expression (any such name can be chosen). Note that the hide operator has not been maintained in the derived expression because the name h that has been created has become public.

A new *agent* is created as a side effect of the execution of an action. The newly created *agent* may share a set of private gate names with its parent *agent*. This is reflected in the transformation of the hide operator into a restrict operator as exemplified by the following derivation:

Example 6

< hide y **in**
(h ? x:t **new** < P[y](x) > ; Q[y,h]) >

restrict g **to** (< P[g](v) > | < Q[g,h] >)

: the initial *agent* begins by receiving a value for x on gate h and then creates a new *agent* < P[g](v) >. Note how the **hide** y **in** _ operator has been transformed into a restriction on the two parallel *agents*. Note also that name g has been chosen as the private gate (any other name different from h would have served the same purpose).

In the case of LOTOS, two actions are synchronisable iff they are identical and their synchronisation results in an identical action. The synchronisation of actions in M-LOTOS needs to be revisited taking into account the following two considerations:

- when two actions from two *agents* synchronise the resulting action is i.
- the synchronisation of gate values is not similar to that of other values. For instance, the offers !h and !g may not synchronise even if h and g are equal, because each of h and g represents the fresh and unique name of a locally created gate. Also, in contrast with values, *spontaneous* gate generation bears no meaning and is not permitted. Thus, the two *agents*: < g?h:gid; *B1*> and <g?k:gid; *B2*> cannot synchronise, and the expression: **hide** h **in** h?x:gid ; B is equivalent to **stop**.

Another important aspect about synchronisation is that it has an effect on the scopes of hide and restrict operators. When a restricted (resp. hidden) gate is passed in a synchronisation, the scope of the restriction (respectively, hide) operator is enlarged to include the receiving expression. Following are two commented examples of derivations involving synchronisation of *agents*.

Example 7

restrict y **to** (< g ! y ; P[y,g] >)
|
< g?x:gid; Q[x] >

restrict h **to** (< P[h, g] > | < Q[h] >)

: the initial *agent-expression* is a parallel composition of two *agent-expressions*, where gate y is restricted to the first one. Gate y is passed, under the name h, to the second *agent*. Note how the scope of the restriction is enlarged to include the recepient of gate h. This phenomenon is called scope extrusion in the π-calculus.

Example 8

< (g !y !0 ; P1[y,g] [] g !y !1 ; P2[y,g]) |[g]|
(g ? u:gid ? w:Nat [w gt 0] ; P3[g]) >
|
< g?x:gid ?z:Nat; Q[x](z) >

< P2[y,g] |[g]| P3[g] >
|
< Q[y](1) >

: this example shows how the constraint oriented style of LOTOS is preserved in M-LOTOS. We have a parallel composition of two *agents*. In the fisrt *agent*, we have a choice between two actions on gate g: on one hand, offering of gate name y and the value 0 and, on the other hand, offering of gate y and 1. The choice is resolved in favour of the second offer by a constraint made in a parallel LOTOS *process*. The resulting action synchronises with yet another *agent*, thus transmitting the values y and 1.

5. FORMAL SEMANTICS OF M-LOTOS

As introduced earlier, we will handle the operational semantics of M-LOTOS in two steps. First, we will consider the LOTOS^M sub-language. The meaning of a LOTOS^M *behaviour-expression* is given by its translation into a special type of Labelled Transition Systems. The translation is defined by a set of SOS rules. Based on this first result, we tackle the operational semantics of M-LOTOS. To begin, we need some auxiliary definitions and notations.

5.1 Auxiliary definitions and notations

- the syntax of actions have been introduced earlier. Both LOTOS^M and M-LOTOS use the same action set (ranged over by α, β, γ, and η) :

$$\alpha ::= i \mid \delta \mid g \mid g\,\omega_1 \ldots \omega_n$$
$$\omega ::= v \mid !h \mid ?h \mid !\underline{h} \mid ?\underline{h}$$

- We define the following functions on visible actions:
  - interaction gate: $ig(\alpha)$ is the interaction gate of α
  - bound inputs: $bi(\alpha) =_{def} \{h \mid ?\,\underline{h}$ occurs in $\alpha\}$
  - bound outputs: $bo(\alpha) =_{def} \{h \mid !\,\underline{h}$ occurs in $\alpha\}$
  - free inputs: $fi(\alpha) =_{def} \{h \mid ?\,h$ occurs in $\alpha\}$
  - free outputs: $fo(\alpha) =_{def} \{h \mid !\,h$ occurs in $\alpha\}$
  - offered gates: $og(\alpha) =_{def} bi(\alpha) \cup bo(\alpha) \cup fi(\alpha) \cup fo(\alpha)$
  - gates: $gt(\alpha) =_{def} ig(\alpha) \cup og(\alpha)$
  - non free outputs: $nfo(\alpha) =_{def} bi(\alpha) \cup bo(\alpha) \cup fi(\alpha)$
- Let $f(\,)$ be any of the above functions, and α_1, α_2, two actions, we will use sometimes use the notation $f(\alpha_1, \alpha_2)$ as a shorthand for $f(\alpha_1) \cup f(\alpha_2)$.
- $fg(F)$ denotes the set of free gates in F (F stands for either an LOTOS^M or M-LOTOS expression). We will use the notation $fg(F, F')$ as a shorthand for $fg(F) \cup fg(F')$.
- Let H denote either a LOTOS^M *behaviour-expression* or a M-LOTOS *agent-expression* or a predicate. We define gate and value substitutions as follows:
  - $\{E_1/x_1, \ldots, E_n/x_n\}H$ is the (simultaneous) substitution, in expression H, of the value-variables $x_1, \ldots, x_n$, by the value expressions $E_1, \ldots, E_n$
  - $\{h_1/g_1, \ldots, h_n/g_n\}H$ is the (simultaneous) substitution, in expression H, of gate names $g_1, \ldots, g_n$ with $h_1, \ldots, h_n$. In this case, change of bound gate names in H may occur in order to avoid confusion.
- we will consider that every value-expression E, of sort t gid, has a value denoted val(E). The set of values of a sort t is denoted dom(t). We also consider (abusively) that val(E) is an expression of sort t.
- $match(\alpha, a, B)$ is a predicate which is true iff the observed action α matches with the syntactical action-denotation a in the context of a *behaviour-expression B*. This predicate is structurally defined below:
  - $match(h, g, B) = \text{true}$ iff $h = g$
  - $match(\alpha\,\omega, a\,off, B) = match(\alpha, a, B)$ and $comp(\omega, off, B)$
  - $comp(v, !E, B) = \text{true}$ iff $v = val(E)$
  - $comp(v, ?x{:}t, B) = \text{true}$ iff $v \in dom(t)$ and $t \neq gid$
  - $comp(!h, !g, B) = \text{true}$ iff $h = g$
  - $comp(?h, ?g{:}gid, B) = \text{true}$ for all h
  - $comp(?\underline{h}, ?g{:}gid, B) = \text{true}$ iff $h \notin fg(B)$

- $\|\alpha / a\|$ is a syntactical substitution defined when we have, for some B, $match(\alpha, a, B) =$ true. Let $\alpha = g\,\omega_1 \ldots \omega_n$ and $a = off_1 \ldots off_n$, each couple $\|\omega_i / off_i\|$ defines an elementary substitution and $\|\alpha / a\|$ is the simultaneous execution of these substitutions. The elementary substitution $\|\omega / off\|$ is defined as follows:
  - $\|v / ?x:t\| = \{v / x\}$ (the substitution of value-variable x by the value v),
  - $\|v / !E\|$ is the identity substitution,
  - $\|!h / !h\|$ is the identity substitution,
  - $\|?h / ?g:gid\|$ is the substitution $\{h / g\}$,
  - $\|?\underline{h} / ?g:gid\|$ is the substitution $\{h / g\}$,
- Let Γ, Γ' be *agent-expressions* or observed actions, α an action and C an *agent-expression*, we define:
  - underscored substitution: $\{\underline{h} / h\}\Gamma$ as follows:
    Γ is a *agent-expression* C: $\{\underline{h} / h\}C$ is the substitution of the free occurrences of gate h by the underscored gate $\underline{h}$,
    Γ is an action α: $\{\underline{h} / h\}\alpha$ is the substitution of occurrences of !h by !$\underline{h}$,
  - *bound gates:* $bg(\Gamma) =_{def} \{h \mid \underline{h} \text{ occurs in } \Gamma\}$, we will sometimes use the notation $bg(\Gamma, \Gamma')$ as a shorthand for $bg(\Gamma) \cup bg(\Gamma')$,
  - *non tagged gates:* $nt(C) =_{def} fg(C) - bg(C)$ which is the set of gates occurring free and non underscored in C.
  - *unbound configuration:* $U(C)$ is the operation of replacing, in *agent* C, the underscored gates by their normal (non underscored) occurrences.
- We define synchronisation on offers and on actions by the following (both operations are denoted with the same symbol $*$. The impossible synchronisation is denoted $\perp$):

  (i) value offers and gate offers do not synchronise

  (ii) value offers synchronise only when equal: $v \neq v' \Rightarrow v * v' = \perp$ and $v * v = v$

  (iii) synchronisation of gate offers is defined in the table:

*	!h	!$\underline{h}$	?h	?$\underline{h}$
!h	!h	$\perp$	!h	!h
!$\underline{h}$	$\perp$	$\perp$	$\perp$	!$\underline{h}$
?h	!h	$\perp$	?h	?h
?$\underline{h}$	!h	!$\underline{h}$	?h	?$\underline{h}$

  (iv) the internal action do not synchronise with any action: $\forall\, \gamma : i * \gamma = \gamma * i = \perp$

  (v) synchronisation of observable actions:

  if n=k and g=h and $\forall\, j : \omega_j * \zeta_j \neq \perp$ then $(g\,\omega_1 \ldots \omega_n) * (h\,\zeta_1 \ldots \zeta_k) = g\,\omega_1 * \zeta_1 \ldots \omega_n * \zeta_n$

  else $(g\,\omega_1 \ldots \omega_n) * (h\,\zeta_1 \ldots \zeta_k) = \perp$

- `resolved(`α`)` is a predicate on actions which is true iff $\alpha \neq \perp$ and no "?" occurs in α.

5.2 Semantics of LOTOS^M

The semantical domain of LOTOS^M *behaviour-expressions* is a set of labelled transition systems the nodes of which are *behaviour-expressions*. The labels represent the actions that can be performed by these expressions and the *agent* that is spwaned along with the actions. More

precisely, as mentioned in the introduction, the derivation of an LOTOS^M expression B is of the form: $B \xrightarrow[C]{\alpha} B'$, where γ is an action and C is a *agent-expression*. Derivations take the simpler form, $B \xrightarrow{\alpha} B'$, when no *agent* is created in the transition. This is typically the case for the LOTOS subset of LOTOS^M. We first define a structural congruence $\equiv$ on LOTOS^M expressions which allows a more concise formulation of the SOS rules:

- If P and Q are alpha equivalent (in the λ-calculus sense) then $P \equiv Q$
- $P \,|[G]|\, Q \equiv Q \,|[G]|\, P$
- $P \,[]\, Q \equiv Q \,[]\, P$
- **hide** $g_1, ..., g_n$ **in** $B \equiv$ **hide** g_1 **in hide** g_2 **in** ... **hide** g_n **in** B
- If $A[g_1, ..., g_k](x_1{:}t_1, ..., x_n{:}t_n) =_{def} B$ then $A[h_1, ..., h_k](E_1, ..., E_n) \equiv B\{h_1/g_1, ..., h_k/g_k\}\{E_1/x_1, ..., E_n/x_n\}$

We now present the set of derivation rules for LOTOS^M:

STRUCTURAL CONGRUENCE

$$\frac{P \equiv Q, \quad Q \xrightarrow[C]{\alpha} Q', \quad P' \equiv Q'}{P \xrightarrow[C]{\alpha} P'}$$

ACTION-PREFIX

OBSERVABLE-ACTION

$$\frac{\texttt{match}(\alpha, a, a\ [Pred]; B) = \text{true}, \quad Pred\,\langle\!|\alpha / a|\!\rangle = \text{true}}{a\ [Pred]; B \xrightarrow{\alpha} B\,\langle\!|\alpha / a|\!\rangle}$$

SILENT-ACTION

$$\frac{}{\mathbf{i}; B \xrightarrow{\mathbf{i}} B}$$

NEW-AGENT

$$\frac{\texttt{match}(\alpha, a, a\ [Pred]\textbf{new}\ C; B) = \text{true}, \quad Pred\,\langle\!|\alpha / a|\!\rangle = \text{true}}{a\ [Pred]\textbf{new}\ C; B \xrightarrow[C\,\langle\!|\alpha / a|\!\rangle]{\alpha} B\,\langle\!|\alpha / a|\!\rangle}$$

The OBSERVABLE-ACTION rule simply states that an action prefix $a\,[Pred]; B$ may perform an action α, provided that `match`$(\alpha, a, a\,[Pred]; B) = \text{true}$ (see the definition of predicate `match` in the previous section) and that $Pred\,\langle\!|\alpha / a|\!\rangle = \text{true}$, where $\langle\!|\alpha / a|\!\rangle$ is the substitution induced by the couple (α, a). The "next" *behaviour-expression* is obtained from B by application of the substitution $\langle\!|\alpha / a|\!\rangle$ The NEW-AGENT rule is the general case for action-prefix. In this rule we are also dealing with an action containing the creation of an *agent*. The substitution $\langle\!|\alpha / a|\!\rangle$ induced by the couple (α, a) is applied to the created *agent*, and the result of this substitution appears in the label of the derived transition.

EXIT

$$\frac{}{\textbf{exit} \xrightarrow{\delta} \textbf{stop}}$$

CHOICE

$$\frac{P \xrightarrow[C]{\gamma} P'}{P \,[]\, Q \xrightarrow[C]{\gamma} P'}$$

HIDE

HIDE-TRANSPARENT

$$\frac{P \xrightarrow[C]{\gamma} P'}{\textbf{hide}\ h\ \textbf{in}\ P \xrightarrow[C]{\gamma} \textbf{hide}\ h\ \textbf{in}\ P'} \quad h \notin gt(\gamma),\ h \notin fg(C) \cup bg(C)$$

EXPORT-NEW-GATE

$$\frac{P \xrightarrow[C]{\gamma} P'}{\textbf{hide}\ h\ \textbf{in}\ P \xrightarrow[C\{\underline{h}/h\}]{\gamma\{\underline{h}/h\}} P'} \quad h \neq ig(\gamma),\ h \notin nfo(\gamma),\ h \in fo(\gamma) \cup nt(C)$$

HIDING-AN-ACTION

$$\frac{P \xrightarrow[C]{\gamma} P'}{\textbf{hide}\ h\ \textbf{in}\ P \xrightarrow[C\{\underline{h}/h\}]{i} \textbf{hide}\ \Delta\ \textbf{in}\ P'} \quad h = ig(\gamma),\ h \notin nfo(\gamma) \cup bg(C),\ resolved(\gamma)$$
where $\Delta = (bo(\gamma) \cup \{h\}) - fg(C)$

The HIDE-TRANSPARENT rule considers the case where the gate to be hidden, h, does not occur in action γ and has no free or bound (underscored) occurrences in the *agent* being created *C*. Thus, hide does not alter the derived action and is re-applied to the "next" *behaviour-expression*.

The EXPORT-NEW-GATE rule considers the case where the gate to be hidden, h, is not the gate of γ, the action of the premise, but h is exported in the transition: either as a free output of γ or as a free non tagged gate of the *agent* being created. Since the gate h is exported, the scope of the hide operator should be enlarged to include the potential receiving *processes* or *agents* and/or the created *agent*. In order to prepare for this enlargement of scope, the occurrences of gate h in *C* and in action γ are tagged (*underscored*). Note also that hide is not applied to the "next" *behaviour-expression*. The tagged occurrences of gate h will be exploited later to regenerate the hide operator and to apply it to the proper scope.

The HIDING-AN-ACTION rule considers the case where the gate to be hidden, h, is the gate of γ, the action of the premise. First, since h is the gate of γ, the action is to be transformed to i. There is however a condition for accepting this silent derivation which is that no gate generation should occur in γ. This is reflected in the predicate `resolved(γ)` which is true when no question marks occur in γ. Second, the gates that are bound outputs in γ and not exported in *C* need to be hidden again now that their scope is defined. Thus, the "next" *behaviour-expression* is **hide** Δ **in** P' where, by abuse of notation, **hide** $\emptyset$ **in** P' denotes P'.

PARALLEL COMPOSITION

PAR-INDEPENDENT

$$\frac{P \xrightarrow[C]{\gamma} P'}{P\ |[G]|\ Q \xrightarrow[C]{\gamma} P'\ |[G]|\ Q} \quad ig(\gamma) \notin G \cup \{\delta\},\ bg(\gamma, C) \cap fg(Q) = \emptyset$$

PAR-SYNCHRONISATION

$$\frac{P \xrightarrow[C]{\gamma} P',\ Q \xrightarrow[D]{\eta} Q'}{P\ |[G]|\ Q \xrightarrow[C\,|\,D]{\gamma * \eta} P'\ |[G]|\ Q'} \quad \begin{array}{l} ig(\gamma) \in G \cup \{\delta\},\ \gamma * \eta \neq \perp, \\ bg(\gamma, C) \cap fg(D, Q) = \emptyset, \\ bg(\eta, D) \cap fg(C, P) = \emptyset \end{array}$$

The PAR-INDEPENDENT rule corresponds to the case where each of the parallel components is free to perform an action independently, on the condition that the gate is not in the synchronisation set of the parallel operator. Note that `bound gates` appearing in γ and C should not be present as free gates of Q in order to avoid confusion.

The PAR-SYNCHRONISATION considers the case where the two parallel components are required to synchronise. This is possible on the condition that the two actions are synchronisable (see the definition of operation $*$ in the previous section). The same condition, as in the previous rule, on the occurrence of `bound gates` is also required.

DISABLE

DISABLE-LEFT	DISABLE-RIGHT	DISABLE-EXIT
$\dfrac{P \xrightarrow[C]{\gamma} P'}{P \,[>\, Q \xrightarrow[C]{\gamma} P' \,[>\, Q}$ $ig(\gamma) \neq \delta,\ bg(\gamma, C) \cap fg(Q) = \emptyset$	$\dfrac{Q \xrightarrow[C]{\gamma} Q'}{P \,[>\, Q \xrightarrow[C]{\gamma} Q'}$	$\dfrac{P \xrightarrow[C]{\delta} P'}{P \,[>\, Q \xrightarrow[C]{\delta} P'}$

ENABLE

ENABLE-LEFT	ENABLE-EXIT
$\dfrac{P \xrightarrow[C]{\gamma} P'}{P >> Q \xrightarrow[C]{\gamma} P' >> Q}$ $ig(\gamma) \neq \delta,\ bg(\gamma, C) \cap fg(Q) = \emptyset$	$\dfrac{P \xrightarrow[C]{\delta} P'}{P >> Q \xrightarrow[C]{i} Q}$

5.3 Semantics of agents

Now we can turn to the formal semantics of *agents*. As seen in the informal introduction to the semantics, an elementary derivation in the transition systems of *agents* is of the form:

$$C \xrightarrow{\alpha} C'$$

where C and C' are *agents* and α is an action. We first define a structural congruence $\equiv$ on *agent-expressions* which allows a more concise formulation of the SOS rules:

- If C and D are alpha equivalent (in the λ-calculus sense) then $C \equiv D$
- $C \mid D \equiv D \mid C$
- If $A[g_1, ..., g_k](x_1{:}t_1, ..., x_n{:}t_n) =_{def} C$ then $A[h_1, ..., h_k](E_1, ..., E_n) \equiv C\,\{h_1/g_1, ..., h_k/g_k\}\{E_1/x_1, ..., E_n/x_n\}$
- **restrict** $g_1, ..., g_n$ **to** $C \equiv$ **restrict** g_1 **to restrict** g_2 **to** ... **restrict** g_n **to** C

We now present the SOS rules for *agents*:

EMBEDDING

WITHOUT-NEW-AGENT	WITH-NEW-AGENT
$\dfrac{P \xrightarrow{\gamma} P'}{<P> \xrightarrow{\gamma} <P'>}$	$\dfrac{P \xrightarrow[C]{\gamma} P'}{<P> \xrightarrow{\gamma}$ **restrict** $bg(C)\text{-}bg(\gamma)$ **to** $(<P'> \mid U(C))$

The rules for embedding provide the semantics of a *agent* from that of the embedded *behaviour-expression*. The WITH-NEW-AGENT rule deals with the case where the *behaviour-expression* performs a transition containing the creation of a new *agent*. Note how the gates $bg(C)\text{-}bg(\gamma)$ are restricted in the resulting *agent*. These gates are being exported privately from P to C, so their scope contains both $< P' >$ and C. This is one of the cases where the hide operator is regenerated, but this time the it is regenerated in the form of a restriction. Note also that if $bg(C)\text{-}bg(\gamma)$ is empty, the expression **restrict** $\varnothing$ **to** $(< P' > | C')$ is an abuse of notation and should be understood as $(< P' > | C')$. In this rule we recover an ordinary (i.e., untagged) *agent-expression* from C by applying the syntactical function $U(C)$. The WITHOUT-NEW-AGENT rule is a special case where there is no creation of new *agents*.

RESTRICTION

RESTRICT-TRANSPARENT

$$\frac{C \xrightarrow{\alpha} C' \qquad h \notin gt(\alpha)}{\textbf{restrict } g \textbf{ to } C \xrightarrow{\alpha} \textbf{restrict } h \textbf{ to } C'}$$

EXPORT-NEW-GATE

$$\frac{C \xrightarrow{\alpha} C' \qquad \begin{array}{l} h \neq ig(\alpha), \\ h \in fo(\alpha), \\ h \notin nfo(\alpha) \end{array}}{\textbf{restrict } h \textbf{ to } C \xrightarrow{\alpha\{\underline{h}/h\}} C'}$$

The RESTRICT-TRANSPARENT rule considers the case where the action of the premise does not contain the restricted gate h. The action is unchanged and the restriction applies to the "next" *agent*.

The EXPORT-NEW-GATE rule considers the case where the interaction gate of the action of the premise is not the restricted gate. But, this time, the action contains an output offer of this restricted gate. The fact that a new name is exported in the action is marked by underscoring the name of this gate in the action. This underscoring is exploited in the synchronisation of *agents* (see the corresponding rule below).

PARALLEL COMPOSITION

PAR-INDEPENDENT

$$\frac{C \xrightarrow{\alpha} C' \qquad bg(\alpha) \cap fg(D) = \varnothing}{C \mid D \xrightarrow{\alpha} C' \mid D}$$

PAR-SYNCHRONISATION

$$\frac{C \xrightarrow{\alpha} C', \; D \xrightarrow{\beta} D' \qquad resolved(\alpha * \beta)}{C \mid D \xrightarrow{i} \textbf{restrict } bo(\alpha,\beta) \textbf{ to } (C' \mid D')}$$

Rule PAR-INDEPENDENT considers the case where an action α is performed by an *agent* in a parallel composition. In this case, the `bound names` of α cannot occur free in the other *agent* in order to avoid confusion of names. Rule PAR-SYNCHRONISATION dictates that two actions α and β synchronise iff not only they are compatible (i.e., $\alpha * \beta \neq \perp$) but also there should be no occurrence of "?" in $\alpha * \beta$ (i.e., `resolved`$(\alpha * \beta)$ is true). Moreover, the resulting action is hidden, thus the bound output names of α and β should be regenerated in a restriction.

5.4 Compatibility of the semantics of LOTOS and M-LOTOS

Any given LOTOS *behaviour-expression B* is also a LOTOS^M expression, and thus has an associated transition system given by the LOTOS^M SOS rules. If we adopt for LOTOS, a semantics based on syntactical substitution of gates, instead of gate relabelling, then the two transition systems associated to *B*, i.e., the LOTOS based one and the LOTOS^M based one, are equal. The proof of this claim can be obtained by recreating the LOTOS set of SOS rules by an inspection and rewriting of those of LOTOS^M whereby conditions and actions are evaluated in the context of the LOTOS special case (no gate passing, no *agent* creation).

5.5 Early bisimulations for M-LOTOS

One can define early and a late bisimulations for M-LOTOS similar to those defined for the π-calculus. We present here briefly the early bisimulation. We define early bisimulations over both M-LOTOS and LOTOS^M.

Definition: a binary relation S_M on *agents* is an early simulation if $C\ S_M\ D$ implies that

If $C \xrightarrow{\alpha} C'$ with α such that $bg(\alpha) \cap fg(C,D) = \emptyset$,
then for some D', $D \xrightarrow{\alpha} D'$ and $C'\ S_M\ D'$

Definition: the relation S_M is an early bisimulation if both S_M and $(S_M)^{-1}$ are early simulations. We define early bisimularity on *agents* $C \sim_M D$ when we have $C\ S_M\ D$ for some early bisimulation S_M.

Definition: a binary relation S_m on LOTOS^M expressions is an early simulation if $P\ S_m\ Q$ implies that:

If $P \xrightarrow[C]{\alpha} P'$ with α and C such that $bg(\alpha, C) \cap fg(P, Q) = \emptyset$ then
for some Q', and some D with $U(D) \sim_M U(C)$ and $bg(D) = bg(C)$: $Q \xrightarrow[D]{\alpha} Q'$ and $P'\ S_m\ Q'$

Definition: the relation S_m is an early bisimulation if both S_m and $(S_m)^{-1}$ are early simulations. We define early bisimularity on *agents* $P \sim_m Q$ to mean that $P\ S_m\ Q$ for some bisimulation S_m.

Properties of $\sim_M$ and $\sim_m$: it is easy to proof that both $\sim_M$ and $\sim_m$ are equivalence relations. Moreover, $\sim_M$ is preserved by all operators on *agents* and $\sim_m$ is preserved by all LOTOS^M operators except action prefix.

For instance, if we take P = x; stop and Q = x; stop ||| (x ; stop |[h]| h; stop) then we have:

$$P \sim_m Q \quad \text{but} \quad \text{g?x:gid; } P \not\sim_m \text{ g?x:gid; } Q$$

The bisimulation $\sim_m$ distinguishes between too many *processes*. For instance, it is interresting to equate the two behaviours: P' = g?x:t; h!x ; **stop** and Q' = g?x:t **new** <h!x; **stop**>; **stop**

Both P' and Q' have a similar behaviour which is to receive a value x on gate g and then transmit it on gate h. They only differ in the way this value is transmitted. In P', x is transmitted by the *process* itself while Q' creates a message <h!x; stop> (in the form of an *agent*) for that purpose. The relation $\sim_m$' defined below possesses this quality. We present it leaving its properties for further investigation.

Definition: $\sim_m$' is an alternative early bisimulation defined by: $P \sim_m' Q$ iff $\langle P \rangle \sim_M \langle Q \rangle$.

6. EXAMPLE: A CONNECTION MANAGER

The following is an example describing a system made of a connections-manager, a network and a user. In the specification, we will concentrate on the connections manager. In M-LOTOS, the general structure of this system is given by:

$$< \text{US}[\,u, \ldots\,] > \;\; | \;\; < \text{CM}[\,u, n\,] > \;\; | \;\; < \text{NW}[\,n, \ldots] >$$

where < US> , < CM > and < NW > are *agents* representing the user, the connection manager and the network, respectively. The connection manager possesses two acting external gates u and n. Gate u takes connection requests from the user, and gate n is the interface to the network. This structure reflects the fact that US, CM and NW are loosely coupled and interact by binary rendez-vous on gates u (between < US > and < CM >) and n (between < CM > and < NW >). The definition of connection manager is given by the following equation:

CM[u, n] := u ?t:gid ?add; **hide** s, r **in** n !s !r !add **new** < SC [t, s, r](empty) > ; CM[u, n]

The behaviour of CM is described by the following (using the syntax of CM):

- action u ?t:gid ?add: this action is a connection request. The user US creates a new name, t, (not shown in the specification of CM) and provides this name to CM on gate u. Thus, gate t is privately shared by the user and CM and is used as an identification of the connection from the user side. CM also receives the address of the remote entity and stores it in add.
- **hide** s,r **in** ... : CM is preparing for the creation of a handler of the connection (called SC) that will be opened. Before spawning this connection handler, CM creates two names, s and r, preparing for a private identification of the network connection to be opened. Gate s is used for the communication of data from the connection handler to the network. Gate r is used for the communication of notifications from the network to the connection handler.
- action n !s !r !add **new** <SC [t, s, r](empty)> : CM sends names s, r and add to the network (NW) using gate n and spawns *agent* < SC>. After this action, CM reinstantiates itself at the initial state. Note that <SC> is an *agent* and thus it runs idependently from <CM>.

Agent < SC > is a single connection handler. It has gate t which is shared privately with the user and which will be used to take data packets from the user. It has two gates, s and r, that are privately shared with the network. Gate s is used to send data packets while gate r is for receiving acknowledgements (positive or negative). < SC> transmits messages keeping copies for retransmission in case of negative acknowledgement. The definition of SC is given by the following equation:

Process SC[t, s, r](M:messages) = TR [t, s] |[s]| RT [s, r] (M) **Endproc**

which is the parallel composition, on gate s, of TR, the transmission *process,* and RT, the retransmission *process.* TR takes messages on gate t and transmits them on gate s. RT is parametrised with the set, M, of messages. Messages are couples (message-id, data). RT takes messages on gate s (as transmitted by TR), takes acknowledgements on gate r, and retransmits negatively acknowledged messages on gate r. TR is defined by the equation:

ProcessTR[t, s] = t ?d:data ; **hide** x **in** s!x !d; TR[t, s] **Endproc**

i.e., TR takes data on gate t (action t?d:data), creates a message-id (**hide** x **in** ...), then sends the message-id and data on gate s (action s!x !d) and then re-instantiates. RT is defined by the equation:

Process RT[s, r](M:messages) :=
s?x:gid ?d:data ; RT [s, r](insert(x, d, M))
[]
r?x:gid !ok ; RT [s, r](remove(x, M))
[]
r?x:gid !nok ; r!x !D(x, M) ; RT [s, r](M)
Endproc

which is a choice between three alternatives (each ending by a reinstantiation of the process):

- taking a new message-id and a new packet of data and then inserting them in M,
- receiving a message-id, x, with a positive acknowledgement, ok, and then removing message x from the set of messages M,
- receiving a message-id, x, with a negative acknowledgement, nok; and then retransmitting message x (extracted from M by function D(x,M)).

It is worth noting in this example that gate s is shared locally by *processes* TR and RT and externally by *process* NW.

7. CONCLUSION

We have presented M-LOTOS an extension to LOTOS which enables the language to capture dynamic reconfiguration features of systems. The extension maintains all the attractive specification styles of LOTOS. One interesting characteristic of M-LOTOS is the flavour of true concurrency that it brings by allowing the representation of direct links between *processes*. We have illustrated how M-LOTOS can be used to specify different types of examples.

Work should progress in three directions. First, it will be useful to investigate the theory underlying M-LOTOS, and more specifically to study the properties of the early and late bisimulations and their associated modal logics. This work can benefit from the body of knowledge that is being developed presently in support of the π-calculus and other mobile calculi. A second issue is to define explicit gate names making the calculus more concrete.

Acknowledgements

The authors wish to thank their colleagues of the ISO E-LOTOS working group for their fruitful comments on a previous version of this document. The present work has been partly supported by CNET (contracts 93PE7200 and 93PE7208).

References

[Astesioano 84]Astesioano E. and Zucca, E., Parametric Channels via Label expressions in CCS, Journal of Theor. Comp. Sciuence, Vol 33, pp45-64, 1984.

[Berry92] G. Berry, G. Boudol: "The chemical abstract machine" — Theoretical Computer Science

vol.96, 1992.

[Bolognesi 89]T. Bolognesi, E. Brinksma, "Introduction to the ISO Specification Language LOTOS", Computer Networks and ISDN Systems 14 (1987), pp25-29.

[Boudol92]G. Boudol: "Asynchrony and the π-calculus" — Research Report no 1702, INRIA, Sophia-Antipolis, France, May 1992.

[Brinksma88]H. Brinksma: "On the design of Extended LOTOS" – PhD thesis, University of Twente, The Netherlands, November 1988.

[Engberg86]U. Engberg, M. Nielsen: " A calculus of communicating systems with label-passing" – Report DAIMI PB-208, Computer Science Department, University of Aarhus, 1986.

[Honda91]K. Honda, M. Tokoro: "An object calculus for asynchronous communication", in Proceedings ECOOP '91, Lecture Notes in Computer Science no 512, Springer Verlag 1991.

[ISO 88] International Standard 8807 - "LOTOS : A Formal Description Technique Based on the Temporal Ordering of Observational Behavior" - 1988

[Levy90] M. Abadi, L. Cardelli, P.L. Curien, J.J. Levy: "Explicit substitutions" — in Proceedings 17th ACM Conf. on Princ. of Prog. Languages (POPL), San Francisco, January 1990.

[Milner89a]R. Milner, J. Parrow, D. Walker: "A Calculus of Mobile Processes: Parts I & II" — LFCS Report ECS-LFCS-89-85, University of Edinburgh, June 1989 & Journal of Information and Computation 100, p 1-77, 1992.

[Milner91]R. Milner, J. Parrow, D. Walker: "Modal Logics for Mobile Processes" — CONCUR'91, LNCS 527, pp 45-60, 1991.

[Milner 90]R. Milner : "Functions as Processes" - INRIA Research Report n 1124, February 1990 - INRIA, Rocquencourt, France & ICALP'90 - International Colloquium on Automata Languages and Programming - July 1990, Warwick (GB).

[Najm93] E. Najm, J.B. Stefani: "An Early Instantiation Semantics for the π-calculus" Internal Report - CNET Issy-lesMoulineaux- Paris.

[Najm91a]E. Najm, J.B. Stefani: "Object-based concurrency: a process-calculus analysis" — In Proceedings TAPSOFT-CAAP '91, Lecture Notes in Computer Science no 494, Springer Verlag, April 1991.

[Najm91b] E. Najm, J.B. Stefani : "Dynamic Configuration in LOTOS". Proceedings of Forte'91 - 4th International Conference on Formal Description Techniques. Sydney, Autralia. November 1991. North Holland.

[Sangiorgi92]D. Sangiorgi: "Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms." Phd thesis, Department of Computer Science, University of Edimburg, 1992.

[Stefani90]J.B. Stefani: "Open Distributed Processing: The Next Target for the Application Of Formal Description Techniques" in the 3rd International Conference On Formal Description Techgniques FORTE 90 – Madrid, Spain (1990).

[Thomsen89]B. Thomsen: "A Calculus of Higher Order Communicating Systems" in Proceedings of 16th Anual Symposium on Principles of Programming Languages, pp 143-154, 1989.

[Vissers90]C.A. Vissers: "FDTs for Open Distributed Systems, a Prospective View" in Proceedings 10th IFIP WG6.1 Workshop on Protocol Specification, Testing and Verification, Ottawa, Canada (1990).

[WD94] ISO/IEC JTC1/SC21 Working Draft on E-LOTOS. Enhancements to LOTOS. July 94.

[Yon87] A. Yonezawa, M. Tokoro, eds : "Object-Oriented Concurrent Systems" - MIT Press 1987.