Using Bisimulation Proof Techniques for the Analysis of Distributed Algorithms [1,2]

Damien Pous, ENS Lyon [*,3]

Université de Lyon, LIP, UCBL, INRIA, CNRS, France.

Abstract

We illustrate the use of recently developped proof techniques for weak bisimulation by analysing a generic framework for the definition of distributed abstract machines based on a message-passing implementation. We first define this framework, and then focus on the algorithm which is used to route messages asynchronously to their destination.

A first version of this algorithm can be analysed using the standard bisimulation up to expansion proof technique. We show that in a second, optimised version, rather complex behaviours appear, for which more sophisticated techniques, relying on termination arguments, are necessary to establish behavioural equivalence.

Key words: weak bisimilarity, up-to techniques, process algebras, distributed abstract machines, forwarders.

Introduction

Recently, many calculi encompassing distribution and mobility have been studied and implemented. Examples include Join [10,11,9], Distributed Pi [16], Nomadic Pict [35], Kells [2], Ambients [5,12,25], Klaim [24], Seals [6]. The algorithms underlying such implementations are generally quite involved and error-prone, so that adequate techniques and methodologies are required in order to prove their correctness, or to reason about possible optimisations.

[*] Damien.Pous@ens-lyon.fr

[1] Research report RR2007-31

[2] An early version of this paper has appeared in the post-proceedings of TGC '06, Lucca, Italy, November 2006 [27].

[3] This work has been supported by the french project ANR ARASSIA "Modularité Dynamique Fiable".

In [31], a distributed abstract machine is defined, to implement the Safe Ambient Calculus [21]: the PAN. The main ingredients in the definition of this machine are *locations* – where local processes are executed – and *forwarders*, that transmit messages between locations. In [17], we defined an optimised version of this machine where useless forwarders can be garbage collected, and fewer messages are transmitted along the network. We proved that the resulting abstract machine is weak barbed bisimilar to the original one. However due to the lack of adequate up-to techniques or compositionality results, this proof is tedious, and cannot easily be used as a basis for further studies.

Motivated by these difficulties, we introduced new up-to techniques for weak bisimulation [26]. Although these techniques improve on previously known techniques, they are developed in a completely abstract setting and their applicability has not yet been evaluated beyond rather simple illustrative examples. In this article, we show how these techniques can be used to develop bisimulation proofs about nontrivial distributed algorithms, and how to do so in a modular way.

A Framework for Distributed Implementations

The first contribution of this work is the definition of a framework to define and reason about distributed implementations of process algebras with mobility. While some parts [27] of this framework form the basis of the PAN abstract machine [31], we got rid of all hypotheses that were related to the implementation of an Ambient-based calculus, so that it should be suitable to execute a wider range of calculi. Notice however that the aim of this article is not to define *the* universal framework for distributed computations, but rather to show how up-to proof techniques can be used in order to analyse algorithms underlying such infrastructures.

We want to represent the execution of a collection of *local processes*, distributed over a set of *locations* – thought of as asynchronous, independent entities. Each local process may:

- evolve by itself, independently from local processes running at distant sites;
- send arbitrary messages to the local processes hosted in other locations;
- receive such messages;
- spawn new local processes, inside new locations;
- migrate to another location, and redirect further messages to that location.

We abstract over the structure of local processes by representing them as the states of an arbitrary labelled transition system (LTS), whose labels correspond to the above primitive interactions. These local processes, together with the set of messages they exchange, form the input of our framework. We

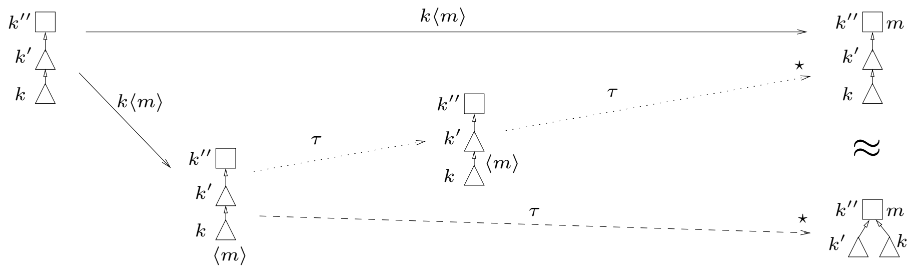


Fig. 1. Sending messages: reference, simple and optimised models.

do not make any assumption on the behaviour of local processes, or about the content of messages. Notably, messages may contain locations or processes, so that π-calculus-like name mobility or higher-order messages are allowed.

We then define another family of LTSs, called *models*, whose labels represent reactions to the previous primitives. Before defining concrete models, we show how to compose any model (an LTS), with the LTS formed by local processes. Intuitively, the *composite LTS* that we obtain executes the local processes concurrently and reacts to the actions they request: sending a message, spawning a location, etc... This approach allows us to separate completely the study of local processes from the study of the models: weak bisimilarity, like other standard behavioural equivalences, is preserved by this construction (Prop. 2.4).

We then focus on several instantiations within the above mentioned family of models. At first, we define a specification, by providing a *reference model* that describes the expected behaviour of the network using a rather coarse grained semantics. This specification makes use of *forwarders* in order to route messages for local processes that migrated from some location to another. A transition of this reference model is depicted on Fig. 1; on the left, there is a net with three locations (k, k' and k''): k and k' are forwarders, whose role is to redirect messages to k' and k'' respectively, and k'' is a *real location*, in the sense that it is intended to host a local process. The emission of a message in the reference model is atomic: a message m sent to k actually reaches the real location k'' by a single transition (the topmost one in the figure).

While the reference model is well-suited to reason about the whole system, it cannot directly be implemented in a distributed way: several locations are involved in order to decide that a single action shall occur (e.g., k, k' and k'' in the figure). We refine this specification into an implementation by defining a *simple model*, where internal – local – actions are used to mimic atomic actions of the reference model: messages are routed along forwarders in an asynchronous way. This behaviour corresponds to the dotted sequence of transitions in Fig. 1: the message is delivered to k, and then routed asynchronously until it reaches k''.

The main drawback of this simple implementation is the persistence of forwarder chains, that slow down the communications between local processes. To address this inefficiency, we introduce an optimisation, inspired from Tarjan's *union-find* algorithm [34]. This optimisation exploits a forwarder relocation mechanism, that contracts forwarder chains. It is expressed as a third *optimised model*, that refines the simple one. This corresponds to the dashed sequence of silent transitions in Fig. 1: in the final net, the chain of forwarders from k to k'' has been contracted. As hinted on this figure, proving that this optimisation is correct will require us to show that the two nets on the right are behaviourally equivalent (weakly bisimilar): the meaning of a net should be independent from the structure of its forwarder trees.

Using "Up-to" Techniques to Validate an Optimisation

The proof of correctness for the optimisation is the main contribution of this work: it allows us to illustrate the use of recent proof techniques [26] to reason about a rather complex, but quite natural, system. We will need the formal setting in order to describe this proof in a useful way; in this introductory section, we just recall the rationale behind "up-to" techniques, and outline those that will be used in the sequel.

The behavioural equivalence we shall consider is *weak bisimilarity* ($\approx$) [23]. Informally, this is the largest symmetric relation $\mathcal{R}$ between the states of an LTS, such that for any label α, the property depicted by the left diagram below holds.

$$\begin{tikzcd}
p \arrow[d, "\alpha"'] & \mathcal{R} & q \arrow[d, Rightarrow, "\widehat{\alpha}"] & & p \arrow[d, "\alpha"'] & \mathcal{R} & q \arrow[d, Rightarrow, "\widehat{\alpha}"] \\
p' & \mathcal{R} & \exists q' & & p' & f(\mathcal{R}) & \exists q'
\end{tikzcd}$$

Hence, in order to show that two states are weakly bisimilar, the usual method is to start with the singleton relation that contains those two states, and then try to enlarge this relation (called the *candidate*), until it satisfies the left diagram above. Notice in particular that due to this diagram, all states accessible from the initial ones will have to be added to the relation at some point. The idea of "up-to" techniques is to alleviate this process by allowing us to check the diagram on the right, where f is a map over binary relations that extends its arguments, so that this diagram is actually easier to satisfy. The correctness of the technique should then ensure that any symmetric relation satisfying this diagram is actually contained in weak bisimilarity. As a side effect, the diagram being easier to satisfy, we need to add fewer pairs to the initial relation, so that we can often work with much smaller candidates.

An intuitive technique is "up to weak bisimilarity", it corresponds to the case where f is the map $\mathcal{R} \mapsto \approx \mathcal{R} \approx$, that allows one to rewrite the states p' and q' using known facts about weak bisimilarity. It is however well known that this technique is not correct [30]: weak bisimilarity being an equivalence that abstracts over silent transitions, when used as an up-to technique, it allows us to "cheat" by cancelling silent transitions. We obtain a valid technique either by restricting its use to *visible* challenges (those where $\alpha \neq \tau$), or by considering *expansion* ($\gtrsim$) [1], a preorder strictly contained in weak bisimilarity that yields a correct technique for weak bisimilarity: the map $R \mapsto \gtrsim \mathcal{R} \lesssim$ [30]. Those two techniques allow one to give more *modular* proofs: we can first show some preliminary expansion or weak bisimilarity results, and then use these in order to obtain a weak bisimulation result. We will actually use these techniques in order to obtain the equivalence of the three models, after having proved some properties of these models.

Another technique, "up to transitivity", seems to be part of the folklore, although less frequently used than the previous ones; it plays a crucial role in our analyses of the simple and optimised models. This up-to technique corresponds to the reflexive transitive closure map ($\mathcal{R} \mapsto \mathcal{R}^\star$); it allows one to decompose a global relation into a much smaller one. For example, both in the simple and in the optimised model, we will have to show that silent transitions ($\xrightarrow{\tau}$) are contained in weak bisimilarity. Unfortunately, in both cases, whenever we try to complete the relation $\xrightarrow{\tau}$ in order to satisfy the left diagram above, we end up adding (at least) its reflexive transitive closure. Being a large relation, checking the diagram directly for this closure is not tractable: while two states related by $\xrightarrow{\tau}$ differ only slightly, which allows us to reason syntactically about their differences, this is no longer the case when they are related by $\xrightarrow{\tau}^\star$. In a sense "up to transitivity" allows one to *localise* the candidate relation: in the previous example, it makes it possible to work with the local relation $\xrightarrow{\tau}$ rather than the global relation $\xrightarrow{\tau}^\star$.

However, "up to transitivity" is not correct for weak bisimilarity (essentially for the same reasons as for "up to weak bisimilarity" – in fact, "up to transitivity" encompasses "up to weak bisimilarity"). There are (at least) three workarounds here: (1) we can restrict the use of the technique to visible challenges – this will not help in our case; (2) we can move to the expansion preorder, that supports this technique; this however requires that the weak bisimulation result we want to prove is actually an expansion result – this will be the case for the simple model but not for the optimised one: we will see that the very "controlled" nature of expansion cannot take into account some mechanisms introduced by the optimisation; or (3) we can obtain the correctness of "up to transitivity" in the weak case by checking an additional termination hypothesis – we will rely on this technique for the optimised model. While techniques (1) and (2) are well-known and straightforward to prove, technique (3) is recent [26] and its proof is much more involved.

Outline. In Sect. 1, we introduce our notations and the standard notions used in the sequel. We define the abstract framework in Sect. 2; Sections 3 and 4 are respectively devoted to the definition of the simple and optimised models, and in each case, to the corresponding correctness proof. We give an overview of the correctness proof, discuss related work, and give directions for future work in Sect. 5.

1 Preliminary Definitions

We assume in this section a set $\mathcal{L}$ of *labels* (or *actions*); this set will be instantiated in different ways in the following sections. Since we will work with weak equivalences, we require that $\mathcal{L}$ contains a distinguished *silent* (or *internal*) action, denoted by τ.

Definition 1.1 (Labelled Transition System, Processes)**.** We call *$\mathcal{L}$-transition system ($\mathcal{L}$-TS)* any pair $\langle \mathcal{P}, \hookrightarrow \rangle$ where $\mathcal{P}$ is an arbitrary set of *states*, and $\hookrightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ is a set of *labelled transitions.* $\mathcal{P}$ is called the *domain* of the $\mathcal{L}$-TS.
An *$\mathcal{L}$-process* is a rooted $\mathcal{L}$-TS: a pair $\langle P, \langle \mathcal{P}, \hookrightarrow \rangle\rangle$ where $\langle \mathcal{P}, \hookrightarrow \rangle$ is a $\mathcal{L}$-TS, and the *root*, P, is a distinguished state of the domain, $\mathcal{P}$.

We write $P \overset{\alpha}{\hookrightarrow} P'$ when $\langle P, \alpha, P' \rangle \in \hookrightarrow$. We shall often denote an $\mathcal{L}$-TS $\langle \mathcal{P}, \hookrightarrow \rangle$ by the set $\hookrightarrow$ of its labelled transitions, when the domain is irrelevant or clear from $\hookrightarrow$. Accordingly, an $\mathcal{L}$-process $\langle P, \langle \mathcal{P}, \hookrightarrow \rangle\rangle$ will be denoted by $\langle P, \hookrightarrow \rangle$, or P, when the $\mathcal{L}$-TS is clear from the context.

Until the end of this section we define several notions that actually depend on $\mathcal{L}$. We do not make this dependency explicit in order to alleviate notations.

We let p, q, r range over $\mathcal{L}$-processes and we let $\mathcal{R}, \mathcal{S}, \mathcal{E}$ range over binary relations between $\mathcal{L}$-processes (simply called *relations* in the sequel). When a relation relates $\mathcal{L}$-processes sharing a unique $\mathcal{L}$-TS, it will be identified with the corresponding binary relation between the states of that $\mathcal{L}$-TS. Let $\mathcal{R}$, $\mathcal{S}$ be two relations; we write $p \mathrel{\mathcal{R}} q$ when $\langle p, q \rangle \in \mathcal{R}$; we denote respectively by $\mathcal{R}^+$, $\mathcal{R}^=$, $\mathcal{R}^\star$ the transitive, reflexive, transitive and reflexive closures of $\mathcal{R}$; the relational composition of $\mathcal{R}$ and $\mathcal{S}$, written $\mathcal{R}\mathcal{S}$, is defined by $\mathcal{R}\mathcal{S} \triangleq \{\langle p, r \rangle \mid p \mathrel{\mathcal{R}} q \text{ and } q \mathrel{\mathcal{S}} r \text{ for some } q\}$; the converse of $\mathcal{R}$ is $\mathcal{R}^{-1} \triangleq \{\langle p, q \rangle \mid q \mathrel{\mathcal{R}} p\}$; $\mathcal{R}$ is symmetric if $\mathcal{R} = \mathcal{R}^{-1}$; we say that $\mathcal{R}$ *contains* $\mathcal{S}$ (alternatively, that $\mathcal{S}$ is contained in $\mathcal{R}$), written $\mathcal{S} \subseteq \mathcal{R}$, if $p \mathrel{\mathcal{S}} q$ implies $p \mathrel{\mathcal{R}} q$. Finally we denote by $\mathcal{I}$ the reflexive relation.

Definition 1.2 (Transition Relations)**.** Any action $\alpha \in \mathcal{L}$ induces a relation, denoted by $\xrightarrow{\alpha}$:

$$\xrightarrow{\alpha} \triangleq \left\{ \langle \langle P, \hookrightarrow \rangle,\ \langle P', \hookrightarrow \rangle \rangle \mid \text{for any } \mathcal{L}\text{-TS} \hookrightarrow \text{ and states } P, P' \text{ s.t. } P \overset{\alpha}{\hookrightarrow}, P' \right\}.$$

Its converse is written using a reversed arrow: $\xleftarrow{\alpha} = (\xrightarrow{\alpha})^{-1}$, and similarly for other forms of arrows, like the following *weak transition relations*:

$$\xrightarrow{\widehat{\alpha}} \triangleq \begin{cases} \xrightarrow{\tau}{}^{=} & \text{if } \alpha = \tau \\ \xrightarrow{\alpha} & \text{otherwise} \end{cases} \qquad \xRightarrow{\alpha} \triangleq \xrightarrow{\tau}{}^{\star} \xrightarrow{\alpha} \xrightarrow{\tau}{}^{\star} \qquad \xRightarrow{\widehat{\alpha}} \triangleq \xrightarrow{\tau}{}^{\star} \xrightarrow{\widehat{\alpha}} \xrightarrow{\tau}{}^{\star}$$

We can remark the following properties: $\xRightarrow{\widehat{\tau}} = \xrightarrow{\tau}{}^{\star}$, $\xRightarrow{\tau} = \xrightarrow{\tau}{}^{+}$ and $\xRightarrow{\widehat{\alpha}} = \xRightarrow{\alpha}$ when $\alpha \neq \tau$ (note in particular the difference between $\xRightarrow{\widehat{\tau}}$ and $\xRightarrow{\tau}$).

Definition 1.3 (Simulation, Bisimulation, Bisimilarity)**.** A relation $\mathcal{R}$ is a *simulation* if for any label $\alpha \in \mathcal{L}$ and $\mathcal{L}$-processes p, q, p', we have:

$$p \mathrel{\mathcal{R}} q \text{ and } p \xrightarrow{\alpha} p' \text{ entail } q \xRightarrow{\widehat{\alpha}} q' \text{ and } p' \mathrel{\mathcal{R}} q' \text{ for some } q'.$$

A *bisimulation* is a symmetric simulation. *Bisimilarity*, denoted by $\approx$, is the union of all bisimulations.

Proposition 1.4. *Bisimilarity is an equivalence relation, it is the greatest bisimulation.*

By defining bisimilarity as a relation between rooted $\mathcal{L}$-TS ($\mathcal{L}$-processes) rather than between the states of a given $\mathcal{L}$-TS, we gain the ability to relate processes which are associated to different transitions systems. This will be useful in order to compare the different versions of our framework. We also define *expansion* ($\gtrsim$), the standard behavioural preorder [1] that leads to the correct "bisimulation up to expansion" technique [30].

Definition 1.5 ((Pre)-Expansion)**.** A *pre-expansion relation* is a relation $\mathcal{R}$ such that for any $\alpha \in \mathcal{L}$, whenever $p \mathrel{\mathcal{R}} q$ we have:

$$\text{if } p \xrightarrow{\alpha} p' \text{ then } q \xrightarrow{\widehat{\alpha}} q' \text{ and } p' \mathrel{\mathcal{R}} q' \text{ for some } q'.$$

An *expansion relation* is a pre-expansion relation $\mathcal{R}$ such that $\mathcal{R}^{-1}$ is a simulation. *Expansion*, denoted by $\gtrsim$, is the union of all expansion relations.

Proposition 1.6. *Expansion is a preorder contained in bisimilarity, it is the greatest expansion relation.*

We have that $\mathcal{R}$ is a simulation iff $\forall \alpha \in \mathcal{L},\ \xleftarrow{\alpha} \mathcal{R} \subseteq \mathcal{R} \xLeftarrow{\widehat{\alpha}}$; accordingly, $\mathcal{R}$ is a pre-expansion iff $\forall \alpha \in \mathcal{L},\ \xleftarrow{\alpha} \mathcal{R} \subseteq \mathcal{R} \xleftarrow{\widehat{\alpha}}$. This kind of concise definitions will be used in sequel to state various up-to techniques involving such kind of "challenges" or "games".

We shall use the notation $\widetilde{x}$ to denote finite multisets of various kind of elements. The empty multiset is denoted by $\emptyset$ and we denote by $x;\widetilde{x}$ (resp. $\widetilde{y};\widetilde{x}$) the addition of an element x (resp. a multiset $\widetilde{y}$) to a multiset $\widetilde{x}$.

Definition 1.7 (Termination)**.** A relation $\mathcal{R}$ *terminates* if there is no infinite sequence $(p_i)_{i\in\mathbb{N}}$ such that $\forall i, p_i \mathrel{\mathcal{R}} p_{i+1}$.

2 A Framework for Distributed Computation

2.1 Interface of the Framework

We let h, k range over a given set $\mathcal{H}$ of *locations*. We assume a set of *elementary messages*, and we let m, n range over finite multisets of elementary messages, that we simply call *messages*; their set is denoted by $\mathcal{M}$. We let α range over a given set $\mathcal{L}$ of *labels* (containing the distinguished element τ). We let P, Q range over a given set $\mathcal{P}$ of *local processes*.

We define the set $\mathcal{L}_p$ of *process labels* with the following syntax:

$$\begin{array}{lll}
\delta ::= & \alpha & \text{(local action)} \\
 & \mid h\langle m\rangle & \text{(message emission)} \\
 & \mid (m) & \text{(message reception)} \\
 & \mid \triangleright h & \text{(migration)} \\
 & \mid \nu h[P] & \text{(location creation)}
\end{array}$$

We assume an $\mathcal{L}_p$-TS with domain $\mathcal{P}$: $\langle \mathcal{P}, \hookrightarrow\rangle$. This is the only $\mathcal{L}_p$-TS we shall consider, so that a local process $P \in \mathcal{P}$ will implicitly represent the $\mathcal{L}_p$-process $\langle P, \hookrightarrow\rangle$.

The process labels correspond to the primitives we alluded to in the introduction: α corresponds to a standalone transition of a local process, $h\langle m\rangle$ to the emission of message m to location h, (m) to the reception of message m, $\triangleright h$ to the migration to some location h, and $\nu h[P]$ to the spawning of a new location h, with local process P.

A *distributed process* is a finite mapping from $\mathcal{H}$ to $\mathcal{P}$, that we write as follows: $h_1 : P_1, \ldots, h_n : P_n$. In this situation, we say that the local process P_i *is hosted at* h_i. The set of distributed processes, ranged over with D, is denoted by $\mathcal{DP}$. In order to represent the execution of distributed processes, we define

$$[\textsc{Loc}_\circ]\ \frac{P \xhookrightarrow{\alpha} P'}{U \circ D, h : P \overset{\alpha}{\longmapsto}_u U \circ D, h : P'} \qquad [\textsc{Int}_\circ]\ \frac{U \xrightarrow{\tau}_u U'}{U \circ D \overset{\tau}{\longmapsto}_u U' \circ D}$$

$$[\textsc{Snd}_\circ]\ \frac{U \xrightarrow{k\langle m\rangle}_u U' \qquad P \xhookrightarrow{k\langle m\rangle} P'}{U \circ D, h : P \overset{\tau}{\longmapsto}_u U' \circ D, h : P'} \qquad [\textsc{Rcv}_\circ]\ \frac{U \xrightarrow{h(m)}_u U' \qquad P \xhookrightarrow{(m)} P'}{U \circ D, h : P \overset{\tau}{\longmapsto}_u U' \circ D, h : P'}$$

$$[\textsc{New}_\circ]\ \frac{U \xrightarrow{\nu k}_u U' \qquad P \xhookrightarrow{\nu k[Q]} P'}{U \circ D, h : P \overset{\tau}{\longmapsto}_u U' \circ D, h : P', k : Q} \qquad [\textsc{Mig}_\circ]\ \frac{U \xrightarrow{h \triangleright k}_u U' \qquad P \xhookrightarrow{\triangleright k} P'}{U \circ D, h : P \overset{\tau}{\longmapsto}_u U' \circ D, \emptyset}$$

Fig. 2. Transitions for the composite $\mathcal{L}$-TS.

an abstract notion of net and we show how to compose such a net with a distributed process.

Definition 2.1 (Models, Nets)**.** A *model* is an $\mathcal{L}_n$-TS, where $\mathcal{L}_n$ is the set of *net labels* defined below; we call *nets* the states of a given model.

$$\begin{array}{lll}
\mu ::= & \tau & \text{(internal action)} \\
 & \mid h\langle m\rangle & \text{(message emission)} \\
 & \mid h(m) & \text{(message reception)} \\
 & \mid h \triangleright k & \text{(migration)} \\
 & \mid \nu h & \text{(location creation)}
\end{array}$$

Net labels closely correspond to process labels: they are actually associated pairwise in order to define the following *composite LTS*, that describes the execution of a distributed process, inside a given model:

Definition 2.2 (Composition of Nets and Distributed Processes)**.** We associate to any model $\langle \mathcal{U}, \rightarrow_u\rangle$ the $\mathcal{L}$-TS $\langle \mathcal{U} \times \mathcal{DP}, \mapsto_u\rangle$ where $\mapsto_u$ is defined by the inference rules given in Fig. 2 (states of the domain are denoted by $U \circ D$).

By rule [$\textsc{Loc}_\circ$], a local process may evolve on its own; accordingly the net may achieve some internal transitions by rule [$\textsc{Int}_\circ$]. A local process hosted at h can send a message to k via rule [$\textsc{Snd}_\circ$], and receive messages by rule [$\textsc{Rcv}_\circ$]. Rule [$\textsc{New}_\circ$] allows a local process hosted at h to spawn a process Q at new location k. Finally, by rule [$\textsc{Mig}_\circ$], a local process hosted at h may migrate to some location k; in that case, the continuation of the local process (P') is lost – we shall comment on this point below.

Remark that since the communication is achieved by unification (rule [$\textsc{Rcv}_\textsc{s}$] requires that the network and the local process agree on the same message m), we actually impose an "early" semantics [32] to local processes. Adapting this rule to the case where local processes have a “late” semantics is possible; however this would require us to ask for a notion of substitution over local processes. Also notice that while we exchange multisets of elementary messages (this facilitates our notations in the sequel), the early semantics allows a local process to decide to receive only single elementary messages.

Remark 2.3 (On the migration mechanism)**.** The rule for migration [$\text{M}\textsc{ig}_\circ$] deserves some explanations: since we are dealing with a migration primitive, we would expect the continuation to be sent to k, for future execution. In order to model directly something like this however, we would have to assume a “merge” operation over processes, to be able to integrate the continuation with the process already hosted at k (this is actually what we did in the conference version of this article [27]).
We claim that the current setting is more general, in the sense that the previous behaviour can be recovered by working at the level of local processes. For example, h could send the continuation to k as a special message $(reg(P'))$ just before migrating; in this case, the process hosted at k has to cooperate, by actually executing the code it receives via such messages. This is illustrated by the hypothetical transitions below, to be read as “assuming the transitions on the left, the composite $\mathcal{L}$-TS will perform the transitions on the right”.

$$\begin{array}{l}
P \xhookrightarrow{k\langle reg(P')\rangle} P_1 \xhookrightarrow{\triangleright k} P_2 \\
Q \xhookrightarrow{(reg(X))} Q \mid X \quad (\forall X) \\
U_1 \xrightarrow{k\langle reg(P')\rangle} U_2 \xrightarrow{k(reg(P'))} U_3 \xrightarrow{h \triangleright k} U_4
\end{array}
\qquad
\begin{array}{rl}
& U_1 \circ h : P,\ k : Q \\
\overset{\tau}{\mapsto} & U_2 \circ h : P_1,\ k : Q \\
\overset{\tau}{\mapsto} & U_3 \circ h : P_1,\ k : (Q \mid P') \\
\overset{\tau}{\mapsto} & U_4 \circ k : (Q \mid P')
\end{array}$$

(notice that we can then refine this scenario into a more secure one, where k would only accept to run cryptographically certified code for example: we are free to implement any protocol at the level of local processes). Also notice that the simpler case of a migration to a *new* location can be achieved directly, by spawning the current state (P) into a new location, and then migrating to that location:

$$\begin{array}{l}
P \xhookrightarrow{\nu k[P]} P_1 \xhookrightarrow{\triangleright k} P_2 \\
U_1 \xrightarrow{\nu k} U_2 \xrightarrow{h \triangleright k} U_3
\end{array}
\qquad
\begin{array}{rl}
& U_1 \circ h : P \\
\overset{\tau}{\mapsto} & U_2 \circ h : P_1,\ k : P \\
\overset{\tau}{\mapsto} & U_3 \circ k : P
\end{array}$$

Finally, remark that migration is *subjective* in our model: local processes decide to migrate by themselves. *Objective* migration mechanisms (like the *passivation* available in the Kell-calculus [2]) may be simulated by using messages to trigger migrations: this is how the *open* primitive of Mobile Ambients was implemented in the initially specialised version of this framework [17].

The main advantage of this approach, where local processes are strongly separated from the network, is that we can study these two entities separately, as expressed by the following proposition:

Proposition 2.4. *Let $\langle U, \rightarrow_u\rangle$ and $\langle V, \rightarrow_v\rangle$ be two nets, let P, Q be two local processes and let D be a distributed process.*

- *If $\langle U, \rightarrow_u\rangle \approx \langle V, \rightarrow_v\rangle$ then $\langle U \circ D, \mapsto_u\rangle \approx \langle V \circ D, \mapsto_v\rangle$.*
- *If $P \approx Q$ then $\langle U \circ D, h : P, \mapsto_u\rangle \approx \langle U \circ D, h : Q, \mapsto_u\rangle$.*

Proof. We first show that for any states U, U', local processes P, P', distributed process D, and location h, we have:

- $U \xrightarrow{\tau}{}^{\star}_{u} U'$ entails $U \circ D \overset{\tau}{\mapsto}{}^{\star}_{u} U' \circ D$, and
- $P \overset{\tau}{\hookrightarrow}{}^{\star} P'$ entails $U \circ D, h : P \overset{\tau}{\mapsto}{}^{\star}_{u} U \circ D, h : P'$.

It follows easily that the two relations below are bisimulations:

$$\mathcal{R}_1 \triangleq \{\langle U \circ D,\ V \circ D\rangle \mid \forall U, V,\ \text{s.t. } \langle U, \rightarrow_u\rangle \approx \langle V, \rightarrow_v\rangle\}\ ,$$
$$\mathcal{R}_2 \triangleq \{\langle U \circ D, h : P,\ U \circ D, h : Q\rangle \mid \forall U, P, Q,\ \text{s.t. } P \approx Q\} \cup \mathcal{I}\ . \qquad \blacksquare$$

We can moreover show that Prop. 2.4 also holds for stronger behavioural preorders or equivalences like expansion ($\gtrsim$) or strong bisimilarity [23].

2.2 Specification of the Expected Behaviour

While the abstract view we have introduced allows us to define several execution models, it does not specify their expected behaviour: only some of all possible models, each corresponding to a possible behaviour, are of interest. In order to fix the expected behaviour we define a "reference model": a model will be "valid" if its states are bisimilar to those of this reference model.

The syntax of *nets* is given in Fig. 3. At first we only consider *reference nets*: the two other extensions of this syntax will be studied later on. We implicitly reason modulo associativity and commutativity of the parallel composition ($|$) which admits $\mathbf{0}$ as a neutral element. Therefore, a reference net is basically a finite multiset where each element is either

- a real location $h[m]$ containing a message m (recall that a message is actually a collection of elementary messages) – in the composite $\mathcal{L}$-TS (Def. 2.2), a local process will be running at such locations;
- or a forwarder $h \triangleright k$ that will redirect any message from h to k.

Intuitively, locations are independent entities that host some "agents" which

reference nets	*simple nets*	*optimised nets*	
$U ::= \mathbf{0}$	$U ::= \mathbf{0}$	$U ::= \mathbf{0}$	(empty net)
$\mid U \mid U$	$\mid U \mid U$	$\mid U \mid U$	(parallel composition)
$\mid h[m]$	$\mid h[m]$	$\mid h[m]$	(real location)
$\mid h \triangleright k$	$\mid h \triangleright k$	$\mid h \triangleright k$	(forwarder)
	$\mid h\langle m \rangle$	$\mid h\langle m \rangle_{\widetilde{k}}$	(pending message)
		$\mid h \not\triangleright k$	(blocked forwarder)
		$\mid h\langle \triangleright k \rangle$	(relocation message)

Fig. 3. Syntax of reference, simple and optimised nets.

have to be uniquely identified. This is what we express using the following well-formedness property:

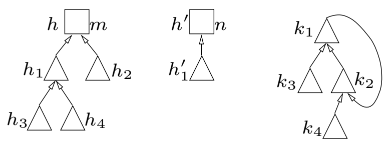
Definition 2.5 (Defined location, well-formedness)**.** A location $h \in \mathcal{H}$ is *defined by* a reference net U if there exist m, V, k such that $U = h[m] \mid V$ or $U = h \triangleright k \mid V$. We denote by $d(U)$ the set of locations defined by U.
A reference net is *well-formed* if it defines every location at most once (i.e., if each location appears at most once as a real location, or as the source of a forwarder), and if for each of its forwarders, $h \triangleright k$, the location k is defined.

Well-formed nets could thus be defined as finite mappings, associating agents to the elements of a finite subset of $\mathcal{H}$. We prefer our explicitly concurrent syntax, which we find both easier to manipulate and more intuitive.

Recall that locations express only the *logical* distribution of processes. Hence, the well-formedness condition does not rule out the case where several locations are thought of as residing *physically* on the same device. Also, unlike in [31,17], real locations (i.e., those hosting local processes in the composite LTS) are not required to be distributed along a tree structure, and there is no constraint on the communication topology. Moreover, since messages may contain locations, π-calculus-like mobility of links is provided by the model.

The definition of well-formedness does not prevent the existence of *forwarder cycles* (we shall explain why below); a generic well-formed net is represented in Fig. 4: it contains two real locations (h and h') to which are pointing two forwarder trees (the h_is and h'_1), and a cycle of forwarders (k_1, k_2), to which points another forwarder tree (k_3, k_4). Using the terminology defined below, the locations h_is admit h as destination, and the k_js are lost. In the sequel, we denote by $\Pi_{i \in I}\, U_i$ the parallel composition of a collection of nets $(U_i)_{i \in I}$.

Definition 2.6 (Destination, lost location)**.** Let U be a well-formed net. The *destination of a location h in U* is the location k if there exist l, m, V and



$$h_3 \rhd h_1 \mid h_4 \rhd h_1 \mid h_1 \rhd h \mid h_2 \rhd h \mid h[m]$$
$$\mid h'_1 \rhd h' \mid h'[n]$$
$$\mid k_4 \rhd k_2 \mid k_3 \rhd k_1 \mid k_2 \rhd k_1 \mid k_1 \rhd k_2$$

Fig. 4. A generic well-formed net.

$$[\textsc{Rcv}]\ \frac{}{h[m;n] \mid U \xrightarrow{h(m)}_{\mathrm{r}} h[n] \mid U} \qquad [\textsc{Snd}]\ \frac{}{h[n] \mid U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} h[m;n] \mid U}$$

$$[\textsc{Kil}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} U}\ h \in l(U) \qquad [\textsc{Fwd}]\ \frac{h \rhd k \mid U \xrightarrow{k\langle m\rangle}_{\mathrm{r}} U'}{h \rhd k \mid U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} U'}$$

$$[\textsc{New}]\ \frac{}{U \xrightarrow{\nu h}_{\mathrm{r}} h[\emptyset] \mid U}\ h \notin d(U) \qquad [\textsc{Mig}]\ \frac{h \rhd k \mid U \xrightarrow{k\langle m\rangle}_{\mathrm{r}} U'}{h[m] \mid U \xrightarrow{h \rhd k}_{\mathrm{r}} U'}$$

Fig. 5. Transitions of reference nets.

$(h_i)_{i\leq l}$ with $h = h_0$, $k = h_l$ such that:

$$U = V \mid \Pi_{i<l}\ h_i \rhd h_{i+1} \mid h_l[m] \ ;$$

it is undefined otherwise. A *lost* location is a defined location whose destination is undefined; we denote by $l(U)$ the set of lost locations of U.

Destination are uniquely determined in well-formed nets; intuitively, a location is lost if it belongs to a cycle of forwarders, or points to such a cycle. We can now define the dynamics of our specification; we describe it below:

Definition 2.7 (Reference Model)**.** The *reference model* is $\langle \mathcal{U}_{\mathrm{r}}, \rightarrow_{\mathrm{r}}\rangle$, where $\mathcal{U}_{\mathrm{r}}$ is the set of well-formed reference nets, and $\rightarrow_{\mathrm{r}}$ is defined by the inference rules of Fig. 5.

The following lemma validates this definition, and allows us to consider only well-formed nets in the sequel.

Lemma 2.8. *Well-formed nets* are preserved by $\rightarrow_r$ *, i.e., if* $U \xrightarrow{\mu}_r U'$ *and* U *is well-formed, then* U' *is well-formed. Hence,* $\langle \mathcal{U}_r, \rightarrow_r\rangle$ *is a model.*

We briefly describe the transitions of reference nets (Fig. 5): when a real location contains some message m, the latter can be received by the local process hosted at that location (rule [Rcv], recall that messages m and n are multiset of elementary messages). When a message is sent to a real location, it is added to the other elementary messages of that location (rule [Snd]); by rule [Kil], any message sent on a lost location just disappears; when a message is sent to a location hosting a forwarder $h \triangleright k$, it is redirected to k by rule [Fwd]. Rule [New] allows one to add a new real location, provided that it is not yet defined. Finally a real location $h[m]$ may migrate to some location k by rule [Mig]; in that case, the real location is replaced by a forwarder $h \triangleright k$, and the message m is routed to k (m contains the elementary messages that reached h, but that the local process hosted at h has not yet consumed).

The following lemma gives a more precise intuition of simple nets' behaviour:

Lemma 2.9. *Let U be a reference net, and let m be an arbitrary message.*

(1) If $U = h \triangleright k \mid V$, then either h and k are lost, or they have the same destination.
(2) $U = k[n] \mid V$ and the destination of h in U is k iff $U \xrightarrow{h\langle m\rangle}_r k[m; n] \mid V$.
(3) The location h is lost in U iff $U \xrightarrow{h\langle m\rangle}_r U$.

This reference model is rather high-level: it does not use internal transitions, and the routing of messages through forwarders is achieved in one atomic step, by successive applications of rule [Fwd]. We can moreover remark that the transitions are deterministic: if $U \xrightarrow{\mu}_{\mathrm{r}} U_1$ and $U \xrightarrow{\mu}_{\mathrm{r}} U_2$, then $U_1 = U_2$. As hinted in the introduction, this makes it easier to reason about the properties of this model, but makes a direct distributed implementation hardly feasible.

Notice that messages can only be sent on defined locations, and that, as a consequence, only migrations to defined locations are accepted. This allows us to keep a simple rule for spawning new locations (rule [New]): we do not need to introduce a scope restriction operator to the syntax, that would require us to reason modulo alpha-renaming of bound locations. On the other hand, messages may be sent to lost locations, and migrations may introduce forwarder cycles (for example, we have $h[m] \xrightarrow{h \triangleright h}_{\mathrm{r}} h \triangleright h$). Although these cases can be considered as erroneous requests of local processes, they cannot easily be detected or prevented in a distributed and asynchronous way (at least at our abstract level, where local processes are seen as "black boxes"). Therefore, by specifying a semantics for those situations, and then proving that the simple and optimised models comply with this semantics, we leave the choice of preventing or supporting such cycles to the designer of local processes (in the conference version of this article [27], we did not handle those cases, so that we had to assume that local processes did not create such cycles along their execution).

$$[\textsc{Rcv}_\mathrm{S}]\ \frac{}{h[m;n] \mid U \xrightarrow{h(m)}_\mathrm{s} h[n] \mid U} \qquad [\textsc{Snd}_\mathrm{S}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_\mathrm{s} h\langle m\rangle \mid U}\ h \in d(U)$$

$$[\textsc{Fwd}_\mathrm{S}]\ \frac{}{h\langle m\rangle \mid h \triangleright k \mid U \xrightarrow{\tau}_\mathrm{s} h \triangleright k \mid k\langle m\rangle \mid U}$$

$$[\textsc{Dst}_\mathrm{S}]\ \frac{}{h\langle m\rangle \mid h[n] \mid U \xrightarrow{\tau}_\mathrm{s} h[m;n] \mid U} \qquad [\textsc{New}_\mathrm{S}]\ \frac{}{U \xrightarrow{\nu h}_\mathrm{s} h[\emptyset] \mid U}\ h \notin d(U)$$

$$[\textsc{Mig}_\mathrm{S}]\ \frac{}{h[m] \mid U \xrightarrow{h \triangleright k}_\mathrm{s} h \triangleright k \mid k\langle m\rangle \mid U'}\ k \in h, d(U)$$

Fig. 6. Transitions of simple nets.

3 A Simple Implementation

We now define another model, called *simple*, which is more tractable from an implementation point of view. We validate this model by proving that simple nets are bisimilar to reference nets.

3.1 Simple Nets

We consider *simple nets*, whose syntax – which is given in Fig. 3 – extends that of reference nets by adding pending messages. They will be used to track the intermediate steps corresponding to the routing of pending messages trough forwarders. We extend the notions of defined location, destination and lost location (Defs. 2.5 and 2.6) to simple nets by ignoring pending messages. Well-formedness is extended as follows:

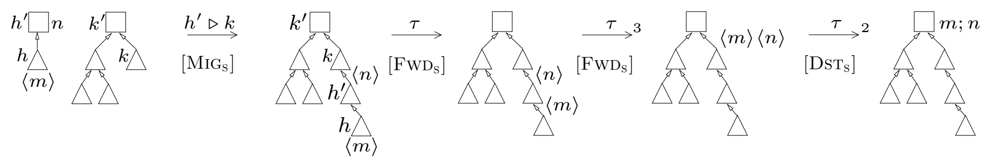
Definition 3.1 (Well-formedness for Simple Nets, Lost Messages)**.** A simple net is *well-formed* if it defines each location at most once, and if for each of its forwarder $h \triangleright k$ or pending message $k\langle m\rangle$, the location k is defined. A pending message $h\langle m\rangle$ is *lost* in a simple net if h is lost in that net.

We can now define the corresponding dynamics:

Definition 3.2 (Simple Model)**.** The *simple model* is $\langle \mathcal{U}_\mathrm{s}, \rightarrow_\mathrm{s}\rangle$, where $\mathcal{U}_\mathrm{s}$ is the set of well-formed simple nets, and $\rightarrow_\mathrm{s}$ is defined in Fig. 6.

Lemma 3.3. *Well-formedness of simple nets is preserved by $\rightarrow_s$; hence, $\langle \mathcal{U}_s, \rightarrow_s\rangle$ is a model.*

Rules [$\textsc{Rcv}_\mathrm{S}$] and [$\textsc{New}_\mathrm{S}$] are unchanged, rule [$\textsc{Snd}_\mathrm{S}$] is simplified: we just add the message to the net, as a pending message. The two silent transition



$$
\begin{array}{rll}
 & h\langle m\rangle \mid h \triangleright h' \mid h[n] \mid k \triangleright k' \mid k'[\emptyset] & \\
\xrightarrow{h' \triangleright k}_{\mathrm{S}} & h\langle m\rangle \mid h \triangleright h' \mid h' \triangleright k \mid k\langle n\rangle \mid k \triangleright k' \mid k'[\emptyset] & [\textsc{Mig}_{\mathrm{S}}] \\
\xrightarrow{\tau}_{\mathrm{S}} & h \triangleright h' \mid h'\langle m\rangle \mid h' \triangleright k \mid k\langle n\rangle \mid k \triangleright k' \mid k'[\emptyset] & [\textsc{Fwd}_{\mathrm{S}}] \\
\xrightarrow{\tau}{}^{3}_{\mathrm{S}} & h \triangleright h' \mid h' \triangleright k \mid k \triangleright k' \mid k'\langle m\rangle \mid k'\langle n\rangle \mid k'[\emptyset] & [\textsc{Fwd}_{\mathrm{S}}] \\
\xrightarrow{\tau}{}^{2}_{\mathrm{S}} & h \triangleright h' \mid h' \triangleright k \mid k \triangleright k' \mid k'[m; n] & [\textsc{Dst}_{\mathrm{S}}]
\end{array}
$$

Fig. 7. Migration and routing of messages in simple nets.

rules are concerned with the routing of messages: rule [$\textsc{Fwd}_{\mathrm{S}}$] defines the behaviour of forwarders: they transmit messages; rule [$\textsc{Dst}_{\mathrm{S}}$] performs the actual reception of a message at a real location. Notice that rule [$\textsc{Mig}_{\mathrm{S}}$], which may seem different from [$\textsc{Mig}$], is actually the same, due to the new definition of $\xrightarrow{h\langle m\rangle}_{\mathrm{S}}$. Finally, there is no rule corresponding to rule [$\textsc{Kil}$]: in simple nets, lost messages persist as garbage.

A sequence of transitions is depicted in Fig. 7, where squares and triangles respectively represent real locations and forwarders; location h' migrates to k and the initial message $h'\langle m\rangle$ is routed to its final destination (k'), as well as the message $k\langle n\rangle$ that resulted from the migration (for the sake of clarity, the three forwarders pointing to k' in the picture are not mentioned in the corresponding formal transitions).

We can remark that the rules defining the simple model have no premise, that each of these rules affects only one location, and that the need for detecting lost locations disappeared along with rule [$\textsc{Kil}$]. Moreover, the side-conditions of rules [$\textsc{Snd}$] and [$\textsc{Mig}$] are easy to enforce: since defined locations cannot disappear, it suffices to check that local processes do not “invent” locations. Those remarks make this model well-suited for a distributed implementation: the programs running at each location can be executed asynchronously.

3.2 Correctness of the Simple Implementation

The main differences between the reference model and the simple one are the use of silent transitions to route pending messages, and the fact that lost messages remain in simple nets, even if they cannot be detected.

The correctness proof goes as follows: we first prove that lost messages can be removed from simple nets without affecting the overall behaviour. This makes it possible to reason about an equivalent model, called *clean*, where nets do not contain lost messages. As far as the simple model is concerned, going through this clean model is not strictly necessary: we include this step because it will be helpful for the correctness proof of the optimised model. We then show that silent transitions of the clean model are contained in weak bisimilarity (this property is commonly called τ-*inertness* [23]). This allows us to prove the equivalence between the reference and the clean model, by reasoning "up to silent transitions".

This model being simple enough, we are able to use up-to techniques based on the expansion preorder ($\gtrsim$). We will see in the following section that this is no longer possible in the optimised model: non standard techniques will be required in order to give the correctness proof for the optimisation.

3.2.1 From simple nets to clean nets.

We start by proving that lost messages can safely be removed:

Lemma 3.4. *Let h be a lost location of a simple net U. For any message m, we have:*

$$U \mid h\langle m\rangle \gtrsim U \ .$$

Proof. We check that $\mathcal{R} \triangleq \{\langle U \mid h\langle m\rangle, U\rangle \mid U \in \mathcal{U}_{\mathrm{s}},\ h \in l(U)\}$ is an expansion relation. We obviously have that $U \xrightarrow{\mu}_{\mathrm{s}} U'$ entails $U \mid h\langle m\rangle \xrightarrow{\mu}_{\mathrm{s}} U' \mid h\langle m\rangle$, and we check that h is still lost in U'. For the other direction, the only non-trivial case is when $U = V \mid h \triangleright k$, $U \mid h\langle m\rangle \xrightarrow{\tau}_{\mathrm{s}} U \mid k\langle m\rangle$. In that case, by Lemma 2.9(1), k is lost in V, so that $U \mid k\langle m\rangle \ \mathcal{R}\ U$: the right-hand-side net does not move. ∎

Definition 3.5 (Clean Nets)**.** A net U is *clean* if it does not contain any lost message. We denote by $\lfloor \mathcal{U}_{\mathrm{s}} \rfloor$ the set of clean simple nets, and we let $\lfloor U \rfloor$ be the clean net obtained from a simple net U by removing all lost messages.

Corollary 3.6. *For any simple net U, we have $U \gtrsim \lfloor U \rfloor$.*

The set of clean nets is not preserved by $\rightarrow_{\mathrm{s}}$: lost messages may be added by rule [SND$_{\mathrm{s}}$], and new forwarder cycles may be created by rule [MIG], so that some "valid" messages may be lost after such transitions. Therefore, in order to obtain a model whose domain is the set of clean nets, we have to remove lost messages after each transition:

Definition 3.7 (Clean Model)**.** The *clean model* is $\langle \lfloor \mathcal{U}_{\mathrm{s}} \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor}\rangle$, where $\rightarrow_{\lfloor \mathrm{s} \rfloor}$

is defined by the following rule:

$$\frac{U \xrightarrow{\mu}_{\mathrm{s}} U'}{U \xrightarrow{\mu}_{\lfloor \mathrm{s} \rfloor} \lfloor U' \rfloor}$$

This model is not realistic from an implementation point of view: it has to detect lost locations after each transition; however, we will only use it to reason about the simple model, by showing that both systems are behaviourally equivalent.

Cor. 3.6 states that any simple net U expands its cleaned form, seen as a state of the simple model. To be able to reason only about clean nets, we actually need to show that the result remains true when we consider the cleaned net as a state of the clean model (Prop. 3.9 below). In order to obtain this result, we will need the following standard up to technique [32]. The idea is that we can use expansion to rewrite the left-hand-side process, when playing the games required by an expansion relation:

Technique 3.8 (Expansion up to Expansion)**.** Let $\mathcal{R}$ be a relation such that for any $\alpha \in \mathcal{L}$, $\xleftarrow{\alpha} \mathcal{R} \subseteq \overset{\widehat{\alpha}}{\longleftarrow} \gtrsim \mathcal{R}$, and $\mathcal{R} \xrightarrow{\alpha} \subseteq \overset{\widehat{\alpha}}{\Rightarrow} \gtrsim \mathcal{R}$.
Then $\gtrsim \mathcal{R}$ is an expansion relation, and $\mathcal{R} \subseteq \gtrsim$.

Proposition 3.9. *For any simple net U, we have*

$$\langle U, \rightarrow_s \rangle \gtrsim \langle \lfloor U \rfloor, \rightarrow_{\lfloor s \rfloor} \rangle \ .$$

Proof. Since for any simple net U, $\langle U, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle \lfloor U \rfloor, \rightarrow_{\mathrm{s}} \rangle$, and $\lfloor U \rfloor$ is a clean net, it suffices to show that $\langle U, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$ for any clean net U.

We apply Technique 3.8 to the following relation:

$$\mathcal{R} \triangleq \left\{ \left\langle \langle U, \rightarrow_{\mathrm{s}} \rangle,\ \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \right\rangle \mid U \in \lfloor \mathcal{U}_{\mathrm{s}} \rfloor \right\} \ .$$

- If $\langle U, \rightarrow_{\mathrm{s}} \rangle \xrightarrow{\mu} \langle U', \rightarrow_{\mathrm{s}} \rangle$, then $\langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \xrightarrow{\mu} \langle \lfloor U' \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$ and we check that $\langle U', \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle \lfloor U' \rfloor, \rightarrow_{\mathrm{s}} \rangle \mathcal{R} \langle \lfloor U' \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$.
- If $\langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \xrightarrow{\mu} \langle U', \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$, then $\langle U, \rightarrow_{\mathrm{s}} \rangle \xrightarrow{\mu} \langle V, \rightarrow_{\mathrm{s}} \rangle$ with $\lfloor V \rfloor = U'$ and we check that $\langle V, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle U', \rightarrow_{\mathrm{s}} \rangle \mathcal{R} \langle U', \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$. ■

3.2.2 From clean nets to reference nets

Now that we have a clean model where any pending message has a destination, we can examine the routing of pending messages, i.e., the silent transitions. We first show that these transitions are convergent:

Lemma 3.10. *The relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$ is terminating and confluent.*

Proof. For any pending message $h\langle m\rangle$ of a clean net U, define its weight as the natural number $l+1$, where l is given by Def. 2.6 (since U is clean, h has a destination in U); then let the size of U be the sum of the weights of all its pending messages. This size strictly decreases along silent transitions, $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$.

Thanks to the well-formedness condition, $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ has no critical pair, so that it is confluent. ■

Notice on the contrary that $\xrightarrow{\tau}_{\mathrm{s}}$ does not terminate, due to the potential presence of pending messages in forwarder cycles. We denote by $U_{\downarrow}$ the normal form w.r.t. $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ of any clean net U. Remark that for any simple net U, $U = \lfloor U \rfloor_{\downarrow}$ if and only if U is actually a reference net.

We now have to prove that silent transitions do not change the behaviour of the system, i.e., that silent transitions are contained in bisimilarity. We actually show a stronger result: they are contained in expansion: this allows us to use the following up-to technique, which is not valid for weak bisimilarity. (Technique 3.11 below actually defines two up-to techniques, the second will be used in Sect. 4.)

Technique 3.11 (Expansion up to Transitivity)**.**
Let $\mathcal{R}$ be a relation such that for any $\alpha \in \mathcal{L}$, $\xleftarrow{\alpha} \mathcal{R} \subseteq \xleftarrow{\widehat{\alpha}} \mathcal{R}^{\star}$.
If $\mathcal{R}^{-1}$ is a simulation or $\mathcal{R}$ is symmetric, then $\mathcal{R} \subseteq \gtrsim$.

Proof. The first hypothesis allows us to show that $\mathcal{R}^{\star}$ is a pre-expansion relation. If $\mathcal{R}^{-1}$ is a simulation, then $(\mathcal{R}^{\star})^{-1} = (\mathcal{R}^{-1})^{\star}$ is a simulation. If $\mathcal{R}$ is symmetric, then we have $(\mathcal{R}^{\star})^{-1} = \mathcal{R}^{\star}$ which is a pre-expansion relation, and hence a simulation. Therefore, $\mathcal{R}^{\star}$ is an expansion relation, and $\mathcal{R} \subseteq \mathcal{R}^{\star} \subseteq \gtrsim$. ■

Lemma 3.12. *The relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$ is contained in expansion.*

Proof. We apply Technique 3.11 to $\mathcal{R} = \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} \cup \mathcal{R}'$, where

$$\mathcal{R}' = \{\langle h\langle m\rangle \mid h\langle n\rangle \mid U,\ h\langle m;n\rangle \mid U\rangle \mid U \in \lfloor \mathcal{U}_{\mathrm{s}} \rfloor,\ h \notin l(U)\}$$

The two cases where we need transitivity are the following visible and silent left-to-right challenges:

- $U = h\langle m\rangle \mid h[n] \mid U_0 \quad \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} \quad h[m;n] \mid U_0 = V$ by rule [DST$_{\mathrm{S}}$], and

$U \xrightarrow{h \rhd k}_{\lfloor \mathrm{s} \rfloor} U' = h\langle m\rangle \mid h \rhd k \mid k\langle n\rangle \mid U_0$ by rule [MIGS]. We have:

$$\begin{array}{lr} V \xrightarrow{h \rhd k}_{\lfloor \mathrm{s} \rfloor} V' = h \rhd k \mid k\langle m; n\rangle \mid U_0 & [\textsc{Mig}_\mathrm{s}] \\ U' \quad \mathcal{R} \quad h \rhd k \mid k\langle m\rangle \mid k\langle n\rangle \mid U_0 \quad \mathcal{R} \quad V' \ ; & [\textsc{Fwd}_\mathrm{s}],(\mathcal{R}') \end{array}$$

- $U = h\langle m\rangle \mid h\langle n\rangle \mid h \rhd k \mid U_0 \quad \mathcal{R}' \quad h\langle m; n\rangle \mid h \rhd k \mid U_0 = V$ and $U \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} U' = h\langle m\rangle \mid h \rhd k \mid k\langle n\rangle \mid U_0$ by rule [FWDS]. In this case we have:

$$\begin{array}{lr} V \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} V' = h \rhd k \mid k\langle m; n\rangle \mid U_0 & [\textsc{Fwd}_\mathrm{s}] \\ U' \quad \mathcal{R} \quad h \rhd k \mid k\langle m\rangle \mid k\langle n\rangle \mid U_0 \quad \mathcal{R} \quad V' . & [\textsc{Fwd}_\mathrm{s}],(\mathcal{R}') \end{array}$$

■

Corollary 3.13. *For any clean net U we have:*

$$\langle U, \rightarrow_{\lfloor s \rfloor}\rangle \gtrsim \langle U_\downarrow, \rightarrow_{\lfloor s \rfloor}\rangle \ .$$

A phenomenon that appears in the proof of the previous lemma allows us to justify the use of the "expansion up to transitivity" technique rather than the more standard "expansion up to expansion" technique (Technique 3.8): if we try to do a direct expansion proof, with $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ as a candidate relation (this is the most natural one), we get stuck in the first case detailed in the proof above: we would like to reason up to the couples of R'. If we could prove that R' is contained in expansion by a preliminary lemma, then we could conclude using the "expansion up to expansion" technique. However, when we try to prove this preliminary lemma, starting with R' itself as a candidate relation, we get stuck in the second case detailed in the proof above: we would like to reason up to $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$... The only way to escape this circular dependency is to prove *jointly* that both R' and $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ are contained in expansion, which requires us to use the "expansion up to transitivity" technique.

We have shown that the routing of messages (i.e., silent transitions) does not affect the behaviour of a net, which will make it possible to reason up to those transitions. However, we still need to show that the routing algorithm actually does something, and that this is what is expected. This is expressed by the following lemma: given a reference net U, if the routing of a message by the reference model leads to a net V, then the clean model can do so, by performing some silent transitions. Conversely, if a pending message is added to U so that we obtain a clean net V, the normalisation of V yields exactly the same simple net as the one we obtain with the reference model.

Lemma 3.14. *Let U be a reference net.*

(1) If $U \xrightarrow{h\langle m\rangle}_r V$, then $U \xrightarrow{h\langle m\rangle}_{\lfloor s \rfloor} \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor s \rfloor} V$.

(2) If $U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} V$, then $U \xrightarrow{h\langle m\rangle}_{r} V_{\downarrow}$.

Proof. In both cases, we distinguish whether h is lost in U or not.

(1) If h is lost in U, then by Lemma 2.9(3) $U = V$, and we have $U \xrightarrow{h\langle m\rangle}_{\lfloor \mathrm{s}\rfloor} \lfloor h\langle m\rangle \mid U\rfloor = \lfloor U\rfloor = U$. Otherwise, h has some destination in U, so that

$$\begin{aligned} U &= U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[n] && \text{(with } h = h_0\text{)} \\ V &= U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m;n], && \text{(Lemma 2.9(2))} \end{aligned}$$

and we check that $U \xrightarrow{h\langle m\rangle}_{\lfloor \mathrm{s}\rfloor} h\langle m\rangle \mid U \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor \mathrm{s}\rfloor} V$, by an induction on l.

(2) If h is lost in U, then $V = \lfloor h\langle m\rangle \mid U\rfloor = U$, and $U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} U$ by Lemma 2.9(3). Otherwise,

$$\begin{aligned} &U = U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[n] && \text{(with } h = h_0\text{)} \\ &U \xrightarrow{h\langle m\rangle}_{\lfloor \mathrm{s}\rfloor} V = h\langle m\rangle \mid U \ \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor \mathrm{s}\rfloor}\ U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m;n] = V'. \end{aligned}$$

where U' and thus V' are necessarily reference nets. By Lemma 2.9(2), $U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} V'$, and since V' is a normal form of V, $V_{\downarrow} = V'$. ■

We now have enough technical devices to give the correctness proof of the implementation w.r.t. the specification. Thanks to the "expansion up to expansion" technique (Technique 3.8), we can work with a very simple candidate relation: syntactical identity between reference nets, equipped with clean and simple transitions on one side, and reference transitions on the other side:

Lemma 3.15. *For any reference net U, we have:*

$$\langle U, \rightarrow_{\lfloor s\rfloor}\rangle \gtrsim \langle U, \rightarrow_{r}\rangle$$

Proof. We apply Technique 3.8 to the following relation:

$$\mathcal{R} = \left\{ \Big\langle \langle U, \rightarrow_{\lfloor \mathrm{s}\rfloor}\rangle,\ \langle U, \rightarrow_{\mathrm{r}}\rangle \Big\rangle \mid U \in \mathcal{U}_{\mathrm{r}} \right\} \ .$$

We consider the different challenges along a label μ:

- μ cannot be τ, since reference nets are in normal form;
- if $\mu = \nu h$ or $\mu = h(m)$, we have $U \xrightarrow{\mu}_{\lfloor \mathrm{s}\rfloor} V$ iff $U \xrightarrow{\mu}_{\mathrm{r}} V$;
- if $\mu = h\langle m\rangle$, we apply Lemma 3.14:
  - · if $U \xrightarrow{h\langle m\rangle}_{\lfloor \mathrm{s}\rfloor} V$, then we have $U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} V_{\downarrow}$ and by Cor. 3.13, we obtain $\langle V, \rightarrow_{\lfloor \mathrm{s}\rfloor}\rangle \gtrsim \langle V_{\downarrow}, \rightarrow_{\lfloor \mathrm{s}\rfloor}\rangle \ \mathcal{R}\ \langle V_{\downarrow}, \rightarrow_{\mathrm{r}}\rangle$ ($V_{\downarrow}$ is a reference net);
  - · if $U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} V$, then we have $U \overset{\widehat{h\langle m\rangle}}{\Longrightarrow}_{\lfloor \mathrm{s}\rfloor} V$ and $\langle V, \rightarrow_{\lfloor \mathrm{s}\rfloor}\rangle \ \mathcal{R}\ \langle V, \rightarrow_{\mathrm{r}}\rangle$.
- The case $\mu = h \triangleright k$ relies on the previous one, and is very similar. ■

Theorem 3.16. *For any simple net U, we have:*

$$\langle U, \rightarrow_s \rangle \succsim \langle \lfloor U \rfloor_\downarrow, \rightarrow_r \rangle \ .$$

Proof. By Prop. 3.9, Cor. 3.13 and Lemma 3.15, we have:

$$\langle U, \rightarrow_\mathrm{s} \rangle \succsim \langle \lfloor U \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \succsim \langle \lfloor U \rfloor_\downarrow, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \succsim \langle \lfloor U \rfloor_\downarrow, \rightarrow_\mathrm{r} \rangle$$

(recall that $\lfloor U \rfloor_\downarrow$ is a reference net). ■

4 Validating an Optimisation

The forwarder chains that are generated along the evolution of a net are the source of inefficiencies. In Fig. 7 for example, any message sent to h will have to go trough three locations before reaching its final destination. In this section, we define an optimised model, that contracts such forwarder chains, and we prove the correctness of this optimisation by showing that simple nets are bisimilar to optimised nets.

4.1 Optimised Nets

The syntax of *optimised nets* is given in Fig. 3; it extends the syntax of simple nets by

- annotating pending messages with a list of locations: $h\langle m \rangle_{\tilde{k}}$;
- introducing *blocked forwarders*: $h \not\triangleright$;
- adding a second kind of messages: *relocation messages* $h\langle \triangleright k \rangle$.

Intuitively, the list that decorates a pending message contains the set of forwarder locations the message did pass through. Messages emitted by the underlying local processes will have an empty list, which will allow us to omit their annotation. Relocation messages are received only by blocked forwarders. Their effect is to redirect such forwarders to a destination closer to the location they indirectly point to.

We extend the notions of defined location and well-formedness to optimised nets, and then define the optimised model:

Definition 4.1 (Defined Locations, Well-formedness for Optimised Nets)**.** A location h is *defined* by an optimised net U if there exist m, V, k such that $U = h[m] \mid V$, $U = h \triangleright k \mid V$ or $U = h \not\triangleright \mid V$. We denote by $d(U)$ the set of

$$[\textsc{Rcv}_{\text{O}}]\ \frac{}{h[m;n] \mid U \xrightarrow{h(m)}_{\text{o}} h[n] \mid U} \qquad [\textsc{Snd}_{\text{O}}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_{\text{o}} h\langle m\rangle_{\emptyset} \mid U}\ h \in d(U)$$

$$[\textsc{New}_{\text{O}}]\ \frac{}{U \xrightarrow{\nu h}_{\text{o}} h[\emptyset] \mid U}\ h \notin d(U)$$

$$[\textsc{Mig}_{\text{O}}]\ \frac{}{h[m] \mid U \xrightarrow{h \triangleright k}_{\text{o}} h \triangleright k \mid k\langle m\rangle \mid U'}\ k \in h, d(U)$$

$$[\textsc{Fwd}_{\text{O}}]\ \frac{}{h\langle m\rangle_{\widetilde{k}} \mid h \triangleright k \mid U \xrightarrow{\tau}_{\text{o}} h\not\triangleright \mid k\langle m\rangle_{h;\widetilde{k}} \mid U}$$

$$[\textsc{Dst}_{\text{O}}]\ \frac{}{h\langle m\rangle_{\widetilde{k}} \mid h[n] \mid U \xrightarrow{\tau}_{\text{o}} h[m;n] \mid \widetilde{k}\langle \triangleright h\rangle \mid U}$$

$$[\textsc{Upd}_{\text{O}}]\ \frac{}{h\langle \triangleright k\rangle \mid h\not\triangleright \mid U \xrightarrow{\tau}_{\text{o}} h \triangleright k \mid U}$$

Fig. 8. Transitions of optimised nets.

locations defined by U.
An optimised net U is *well-formed* if:

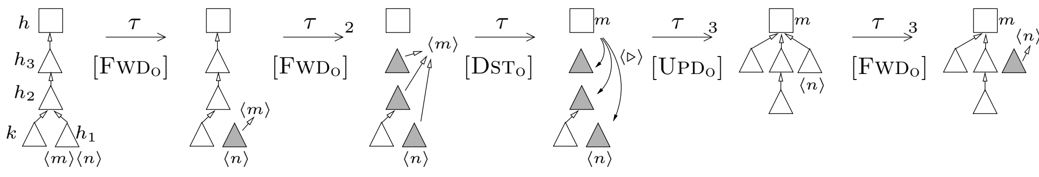
(1) any location $h \in \mathcal{H}$ appears at most once as a real location ($h[m]$), as the source of a forwarder ($h \triangleright k$) in U, or as the source of a blocked forwarder ($h\not\triangleright$); and for any forwarder $h \triangleright k$, pending message $k\langle m\rangle$, or relocation message $h\langle \triangleright k\rangle$, the location k is defined.
(2) for any blocked forwarder $h\not\triangleright$, h appears exactly once in the annotation of a pending message ($k\langle m\rangle_{\widetilde{h}}$, with $h \in \widetilde{h}$), or as the destination of a relocation message ($h\langle \triangleright k\rangle$);
(3) any location registered in a pending message or appearing as the target of a relocation message hosts a blocked forwarder.

Definition 4.2 (Optimised Model)**.** The *optimised model* is $\langle \mathcal{O}, \rightarrow_{\text{o}}\rangle$, where $\mathcal{O}$ is the set of well-formed optimised nets, and $\rightarrow_{\text{o}}$ is defined in Fig. 8.

Proposition 4.3. *Any simple well-formed net in the sense of Def. 2.5 is well-formed in the sense of Def. 4.1. Well-formed optimised nets are preserved by* $\rightarrow_o$*, so that* $\langle \mathcal{O}, \rightarrow_o\rangle$ *is a model.*

This allows us to consider only well-formed nets in the sequel.

We describe the rules below; in comparison to the simple model, "visible" rules [Rcv], [Snd], [Mig] and [New] are left substantially unchanged (except for the empty list decorating pending messages in rule [$\textsc{Snd}_{\text{O}}$]), the two "silent" rules [$\textsc{Fwd}_{\text{S}}$] and [$\textsc{Dst}_{\text{S}}$] are updated and the rule [$\textsc{Upd}_{\text{O}}$] is new. In rule [$\textsc{Dst}_{\text{O}}$], we use $\widetilde{h}\langle \triangleright k\rangle$ to denote $\Pi_{h \in \widetilde{h}}\ h\langle \triangleright k\rangle$. In the sequel, we shall



h
h_3
h_2
k
h_1
$\langle m\rangle\langle n\rangle$
$\xrightarrow{\tau}$ [Fwd$_\mathrm{O}$]
$\langle m\rangle$
$\langle n\rangle$
$\xrightarrow{\tau}^2$ [Fwd$_\mathrm{O}$]
$\langle m\rangle$
$\langle n\rangle$
$\xrightarrow{\tau}$ [Dst$_\mathrm{O}$]
m
$\langle\triangleright\rangle$
$\langle n\rangle$
$\xrightarrow{\tau}^3$ [Upd$_\mathrm{O}$]
m
$\langle n\rangle$
$\xrightarrow{\tau}^3$ [Fwd$_\mathrm{O}$]
m
$\langle n\rangle$

$$
\begin{array}{lll}
 & h_1\langle n\rangle \mid h_1\langle m\rangle \mid h_1 \triangleright h_2 \mid h_2 \triangleright h_3 \mid h_3 \triangleright h \mid h[\emptyset] \mid k \triangleright h_2 & \\
\xrightarrow{\tau}_{\lfloor o\rfloor} & h_1\langle n\rangle \mid h_1\ntriangleright \mid h_2\langle m\rangle_{h_1} \mid h_2 \triangleright h_3 \mid h_3 \triangleright h \mid h[\emptyset] \mid k \triangleright h_2 & [\mathrm{Fwd_O}] \\
\xrightarrow{\tau}{}^{2}_{\lfloor o\rfloor} & h_1\langle n\rangle \mid h_1\ntriangleright \mid h_2\ntriangleright \mid h_3\ntriangleright \mid h\langle m\rangle_{h_1,h_2,h_3} \mid h[\emptyset] \mid k \triangleright h_2 & [\mathrm{Fwd_O}] \\
\xrightarrow{\tau}_{\lfloor o\rfloor} & h_1\langle n\rangle \mid h_1\ntriangleright \mid h_2\ntriangleright \mid h_3\ntriangleright \mid h_1\langle\triangleright h\rangle \mid h_2\langle\triangleright h\rangle \mid h_3\langle\triangleright h\rangle \mid h[m] \mid k \triangleright h_2 & [\mathrm{Dst_O}] \\
\xrightarrow{\tau}{}^{3}_{\lfloor o\rfloor} & h_1\langle n\rangle \mid h_1 \triangleright h \mid h_2 \triangleright h \mid h_3 \triangleright h \mid h[m] \mid k \triangleright h_2 & [\mathrm{Upd_O}] \\
\xrightarrow{\tau}_{\lfloor o\rfloor} & h_1\ntriangleright \mid h\langle n\rangle_{h_1} \mid h_2 \triangleright h \mid h_3 \triangleright h \mid h[m] \mid k \triangleright h_2 & [\mathrm{Fwd_O}]
\end{array}
$$

Fig. 9. Optimised Forwarder Behaviour

furthermore use $\widetilde{h} \triangleright k$ and $\widetilde{h}\ntriangleright$ to respectively denote $\Pi_{h\in\widetilde{h}}\, h \triangleright k$ and $\Pi_{h\in\widetilde{h}}\, h\ntriangleright$.

When a forwarder transmits a message (rule [Fwd$_\mathrm{O}$]), it registers its location and enters a *blocked* state so that it will temporarily block further potential messages. Upon reception at the final location, a relocation message is broadcasted to the locations registered in the message (rule [Dst$_\mathrm{O}$]). The blocked forwarders located at these locations will then update their destination accordingly (rule [Upd$_\mathrm{O}$]). This behaviour is illustrated in Fig. 9, where grey triangles correspond to blocked forwarders. Notice that forwarders have to block until they receive the relocation message: otherwise, a timestamps mechanism would be required, so that a forwarder can cleverly choose between two possibly distinct relocation messages.

Remark 4.4 (Further possible optimisations)**.** We could go beyond the optimisation presented here: for example, pending messages waiting behind a blocked forwarder could be packed together so that they can be sent at once, when the blocked forwarder gets relocated:

$$[\mathrm{Pck_O}]\ \frac{}{h\langle m\rangle_{\widetilde{h}} \mid h\langle n\rangle_{\widetilde{k}} \mid h\ntriangleright \mid U \xrightarrow{\tau}_{\mathrm{o}} h\langle m;n\rangle_{\widetilde{h};\widetilde{k}} \mid h\ntriangleright \mid U}$$

We could also decompose rule [Dst$_\mathrm{O}$], in order to make the broadcast of the relocation message explicit and asynchronous:

$$[\mathrm{Dst_O}]\ \frac{}{h\langle m\rangle_{k;\widetilde{k}} \mid h[n] \mid U \xrightarrow{\tau}_{\mathrm{o}} h\langle m\rangle_{\widetilde{k}} \mid h[n] \mid k\langle\triangleright h\rangle \mid U}$$

$$[\mathrm{Dst'_O}]\ \frac{}{h\langle m\rangle_{\emptyset} \mid h[n] \mid U \xrightarrow{\tau}_{\mathrm{o}} h[m;n] \mid U}$$

However, rather than finding the best implementation of the specification, the aim of this article is to give a methodology for the analysis of such distributed systems. Therefore, we prefer sticking to our rather simple optimisation, so that hopefully our methodology will not get lost into technical details.

4.2 Correctness of the Optimised Model

We now prove the correctness of the optimisation. Like previously, we first show that we can remove lost messages from optimised nets, and work in a "clean and optimised" model (Sect. 4.2.1). The most delicate part of the proof consists in proving that silent transitions are contained in weak bisimilarity (Sect. 4.2.2), which finally allows us to relate simple nets and optimised nets, by reasoning modulo silent transitions (Sect. 4.2.3).

4.2.1 Handling forwarder cycles

We start by adapting the notion of destination to optimised nets, in order to take blocked forwarders into account:

Definition 4.5 (Destination in Optimised)**.** A location h admits k as *destination* in an optimised net U if there exist l, m, V and $(h_i)_{i \leq l}$ with $h = h_0$, $k = h_l$ and such that $U = \Pi_{i<n}\, F_i \mid h_l[m] \mid V$, where for all $i < l$, F_i is either:

- a forwarder: $h_i \triangleright h_{i+1}$, or
- a blocked forwarder, together with a relocation message: $h_i \ntriangleright \mid h_i\langle \triangleright h_{i+1}\rangle$, or
- a blocked forwarder whose location is registered in a message blocking some other forwarders: $h_i \ntriangleright \mid \widetilde{k} \ntriangleright \mid h_{i+1}\langle n\rangle_{h_i;\widetilde{k}}$.

The notions of lost messages and clean nets are extended accordingly; we have the following properties:

Lemma 4.6. *(1) Destinations are uniquely determined in any well-formed optimised net.*
(2) In each of the following nets, h is lost if and only if k is lost:

$$(a)\ h \triangleright k \mid U\,, \qquad (b)\ h \ntriangleright \mid h\langle \triangleright k\rangle \mid U\,, \qquad (c)\ h\langle m\rangle_{k,\widetilde{k}} \mid U\,.$$

Due to blocked forwarders, we can no longer directly remove lost messages. We introduce to this end the following relation:

Definition 4.7. We call *cleaning relation* the following relation over $\mathcal{O}$:

$$\mathcal{E} \triangleq \left\{ \left\langle U \mid \widetilde{h} \ntriangleright \mid h\langle m\rangle_{\widetilde{h}}\ ,\ U \mid \widetilde{h} \triangleright h \right\rangle \;\middle|\; h \in l(U \mid \widetilde{h} \ntriangleright \mid h\langle m\rangle_{\widetilde{h}}) \right\}\ .$$

Notice that V is well-formed whenever $U \mathcal{E} V$ and U is well-formed, and that $\mathcal{E}$ preserves destinations and lost locations. Although it would be sufficient to show that $\mathcal{E}$ is contained in weak bisimilarity, we show that it is contained in expansion, which makes it possible to use the "expansion up to transitivity" technique (Technique 3.11) [4]:

Lemma 4.8. *The cleaning relation is contained in expansion.*

Proof. Let $\mathcal{E}'$ be the symmetric closure of the following relation:

$$\left\{\left\langle U,\ U_0 \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h \right\rangle \,\middle|\, U = U_0 \mid \widetilde{h}\not\triangleright \mid h\langle m\rangle_{\widetilde{h}} \text{ and } h, (k_h)_{h\in\widetilde{h}} \in l(U)\right\}$$

$\mathcal{E}$ is clearly contained in $\mathcal{E}'$; and $\mathcal{E}'$ preserves well-formed nets, destinations and lost locations. We show that $\mathcal{E}'$ is a candidate to Technique 3.11. Since $\mathcal{E}'$ is symmetric, it suffices to show that this is a "pre-expansion up to transitivity". The well-formedness condition allows us to rule out a lot of cases, the remaining interesting cases are the following:

- $U = U_0 \mid \widetilde{h}\not\triangleright \mid h\langle m\rangle_{\widetilde{h}} \mid h \triangleright k \quad \mathcal{E}' \quad V = U_0 \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h \mid h \triangleright k$ and $U \xrightarrow{\tau}_{\mathrm{o}} U' = U_0 \mid (h;\widetilde{h})\not\triangleright \mid k\langle m\rangle_{h;\widetilde{h}}$ by rule [Fwd$_\mathrm{O}$]. We just check that $U' \mathcal{E}' V$: the right-hand-side process does not move.
- $U = U_0 \mid \widetilde{k}\not\triangleright \mid k\langle n\rangle_{\widetilde{k}} \mid (k;\widetilde{h})\not\triangleright \mid h\langle m\rangle_{k;\widetilde{h}}$
  $\mathcal{E}' \quad V = U_0 \mid \widetilde{k}\not\triangleright \mid k\langle n\rangle_{\widetilde{k}} \mid k \triangleright k' \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h$
  and $V \xrightarrow{\tau}_{\mathrm{o}} V' = U_0 \mid (k;\widetilde{k})\not\triangleright \mid k'\langle n\rangle_{k;\widetilde{k}} \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h$ by rule [Fwd$_\mathrm{O}$]. In this case, we have $V \ \mathcal{E}'^2 \ V'$:

  $$V \quad \mathcal{E} \quad U_0 \mid (k;\widetilde{k}) \triangleright k' \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h \quad \mathcal{E}' \quad V' \ .$$

  Hence the left-hand-side process (U) does not move, and we apply $\mathcal{E}'$ three times in order to relate U and V': $U \ \mathcal{E}' \ V \ \mathcal{E}'^2 \ V'$. ■

The cleaning relation is confluent and terminating; we denote by $\lfloor U \rfloor$ the normal form of an optimised net U w.r.t. $\mathcal{E}$ (on simple nets, this function coincides with that defined in Sect. 3.2.1). Like previously, we use this "cleaning function" to define a model over clean optimised nets (whose set will be denoted by $\lfloor \mathcal{O} \rfloor$), which is equivalent to the initial optimised one:

Definition 4.9 (Clean and Optimised Model)**.** The *clean optimised model* is

[4] More precisely, we show that $\mathcal{E}$ is contained in *bi-expansion* [8], the even stronger behavioural equivalence obtained by considering pre-expansion games on both sides: we need to reason up-to transitivity in both directions, and expansion supports this technique only in the left-to-right part of the game.

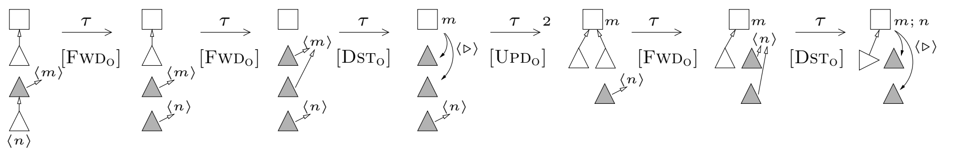


Fig. 10. Unblocking a forwarder in order to route a given message.

$\langle\lfloor\mathcal{O}\rfloor, \rightarrow_{\lfloor o\rfloor}\rangle$, where $\rightarrow_{\lfloor o\rfloor}$ is defined by the following rule:

$$\frac{U \xrightarrow{\mu}_{o} U'}{U \xrightarrow{\mu}_{\lfloor o\rfloor} \lfloor U'\rfloor}$$

Proposition 4.10. *For any optimised net U, we have:*

$$\langle U, \rightarrow_o\rangle \gtrsim \langle\lfloor U\rfloor, \rightarrow_{\lfloor o\rfloor}\rangle \ .$$

Proof. Identical to the proof of Prop. 3.9, using Lemma 4.8. ■

4.2.2 Validating the Routing Algorithm

In order to reason modulo silent transitions, we need to show that they are contained in weak bisimilarity. Unlike in section 3.2.2 however, we cannot rely on expansion-based up-to techniques to prove it: optimised silent transitions are not contained in expansion. This comes from the race conditions introduced by the blocking behaviour of forwarders: for example, in Figs. 9 and 10, the message n has to wait for the arrival of m. The very "controlled" nature of expansion – the right-hand-side process has to be as fast as the left-hand-side process, at each step – cannot take into account the fact that n is closer to its destination at the end.

We will actually use the following technique, from [26], that relies on a termination argument in order to allow the "up to transitivity" technique:

Technique 4.11 (Simulation up to Transitivity, Constrained)**.**
Let $\mathcal{R}$ be a relation such that $\forall\alpha \in \mathcal{L}$, $\xleftarrow{\alpha} \mathcal{R} \subseteq \mathcal{R}^{\star} \overset{\widehat{\alpha}}{\Leftarrow}$.
If $\mathcal{R}^{+}\overset{\tau}{\Rightarrow}$ terminates, then $\mathcal{R}^{\star}$ is a simulation.

Since we work with clean nets, any pending message has a destination; we first prove a lemma that allows us to route these messages to their destination. While this was almost trivial in the simple model (Lemma 3.14(1)), here, as depicted on Fig. 10, in order to route a given message m to its destination, we first have to route any message that blocks some forwarder between the message and its destination. As a side effect, this involves some reconfiguration of the forwarders tree.

Lemma 4.12. *Let $U = V \mid h_0\langle m_0\rangle_{\widetilde{h}}$ be a (clean) optimised net. By Def. 4.5, $U = V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid \Pi_{i<l}F_i \mid h_n[n]$, and we have:*

$$U \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} \; V' \mid \widetilde{h}\langle \triangleright h_l\rangle \mid \Pi_{i<l} h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l$$

where n and $\widetilde{k}$ are the messages and forwarder locations collected in $\Pi_{i<l}F_i$.

Proof. We proceed by induction over l: if $l = 0$ we simply apply rule [DST$_\text{O}$], otherwise we reason by case analysis on the shape of F_0:

- a simple forwarder: $h_0 \triangleright h_1$: we transmit the message with rule [FWD$_\text{O}$], apply the induction hypothesis (IH), and relocate the forwarder using rule [UPD$_\text{O}$]:

$$\begin{aligned}
U &\xrightarrow{\tau}_{\lfloor \text{o} \rfloor} V' \mid h_0 \not\triangleright \mid h_1\langle m_0\rangle_{h_0,\widetilde{h}} \mid \Pi_{0<i<l}F_i \mid h_l[n] && [\text{FWD}_\text{O}]\\
&\overset{\tau}{\Rightarrow}_{\lfloor \text{o} \rfloor} V' \mid h_0 \not\triangleright \mid (h_0, \widetilde{h})\langle \triangleright h_l\rangle \mid \Pi_{0<i<l} h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l && (\text{IH})\\
&\xrightarrow{\tau}_{\lfloor \text{o} \rfloor} V' \mid \widetilde{h} \triangleright h_l \mid h_0 \triangleright h_l \mid \Pi_{i<l} h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l && [\text{UPD}_\text{O}]
\end{aligned}$$

- a blocked forwarder with a relocation message: $h_0 \not\triangleright \mid h_0\langle \triangleright h_1\rangle$: we relocate the forwarder with rule [UPD$_\text{O}$] and fall back into the previous case.
- $(h_0; \widetilde{k}_1) \not\triangleright \mid h_1\langle m_1\rangle_{h_0,\widetilde{k}_1}$: we apply the induction hypothesis to the message at h_1, and transmit the initial message trough the relocated forwarder:

$$\begin{aligned}
U &= V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1) \not\triangleright \mid h_1\langle m_1\rangle_{(h_0;\widetilde{k}_1)} \mid \Pi_{0<i<l}F_i \mid h_l[n]\\
&\overset{\tau}{\Rightarrow}_{\lfloor \text{o} \rfloor} V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1) \not\triangleright \mid (h_0; \widetilde{k}_1)\langle \triangleright h_l\rangle \mid \Pi_{0<i<l} h_i \triangleright h_l\\
&\qquad \mid h_l[m_1; m; n] \mid \widetilde{k} \triangleright h_l && (\text{IH})\\
&\overset{\tau}{\Rightarrow}_{\lfloor \text{o} \rfloor} V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1) \triangleright h_l \mid \Pi_{0<i<l} h_i \triangleright h_l \mid h_l[m_1; m; n] \mid \widetilde{k} \triangleright h_l\\
& && [\text{UPD}_\text{O}]\\
&\xrightarrow{\tau}{}^{3}_{\lfloor \text{o} \rfloor} V' \mid \widetilde{h}\langle \triangleright h_l\rangle \mid \Pi_{i<l} h_i \triangleright h_l \mid h_l[m_0; m_1; m; n] \mid (h_0; \widetilde{k}_1; \widetilde{k}) \triangleright h_l\\
& && [\text{FWD}_\text{O},\text{DST}_\text{O},\text{UPD}_\text{O}]
\end{aligned}$$

■

Despite these modifications of forwarder trees, the routing of messages is convergent. Notice that this is not true in the presence of pending messages whose location is in a forwarder cycle: in that case, the way the cycle gets blocked may depend on the order of the routing steps. We will prove termination below (Lemma 4.17); we first show local confluence:

Lemma 4.13. $\xrightarrow{\tau}_{\lfloor o \rfloor}$ *is locally confluent.*

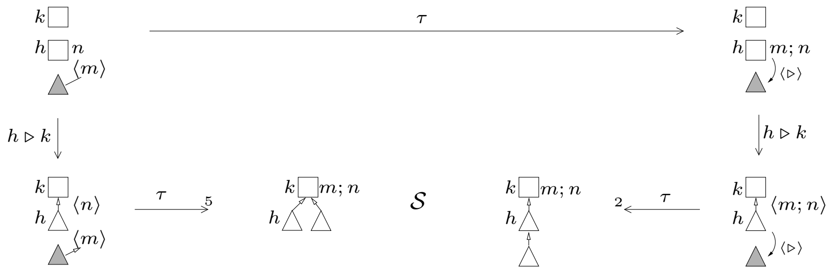


Fig. 11. Commutation of visible and silent transitions, modulo the swapping relation.

Proof. By well-formedness, the only critical pair is the following:

$$\begin{aligned}
&U = V \mid h\langle m\rangle_{\widetilde{h}} \mid h\langle n\rangle_{\widetilde{k}} \mid h \triangleright k \\
&U \xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor} V \mid h\langle m\rangle_{\widetilde{h}} \mid h\not\triangleright \mid k\langle n\rangle_{h,\widetilde{k}} = U_1 && [\textsc{Fwd}_{\mathrm{O}}] \\
&U \xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor} V \mid h\langle n\rangle_{\widetilde{k}} \mid h\not\triangleright \mid k\langle m\rangle_{h,\widetilde{h}} = U_2 && [\textsc{Fwd}_{\mathrm{O}}]
\end{aligned}$$

By using Lemma 4.12 on U_1, we can route the message n to some location k':

$$\begin{aligned}
U_1 &\overset{\widehat{\tau}}{\Rightarrow}_{\lfloor\mathrm{o}\rfloor} V' \mid h\langle m\rangle_{\widetilde{h}} \mid h\not\triangleright \mid (h,\widetilde{k})\langle\triangleright k'\rangle \mid k'[n;m'] && \text{(Lemma 4.12)} \\
&\xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor} V' \mid h\langle m\rangle_{\widetilde{h}} \mid h \triangleright k' \mid \widetilde{k}\langle\triangleright k'\rangle \mid k'[n;m'] && [\textsc{Upd}_{\mathrm{O}}] \\
&\xrightarrow{\tau}{}^{3}_{\lfloor\mathrm{o}\rfloor}\; V' \mid (\widetilde{k},\widetilde{h})\langle\triangleright k'\rangle \mid h \triangleright k' \mid k'[m;n;m'] = U' && [\textsc{Fwd}_{\mathrm{O}},\textsc{Dst}_{\mathrm{O}},\textsc{Upd}_{\mathrm{O}}]
\end{aligned}$$

The same reasoning about U_2 leads to $U_2 \overset{\tau}{\Rightarrow}_{\lfloor\mathrm{o}\rfloor} U'$. ■

Due to forwarders relocations, $\xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor}$ does not commute with visible actions. A counter-example is depicted on Fig. 4.2.2, where the relation $\mathcal{S}$ which is required to close the diagram is defined below. This relation will allow us to reorganise step by step the forwarder structure of a net: while in the case of Fig. 4.2.2, a single application of this relation was sufficient to close the diagram, several applications of this relation will be required in the general case (Lemmas 4.15 and 4.16 below).

Definition 4.14. We denote by $\mathcal{S}$ the *swapping relation*, defined as the symmetric closure of the following relation over $\lfloor\mathcal{O}\rfloor$:

$$\{\langle h \triangleright h' \mid h' \triangleright k \mid U,\ h \triangleright k \mid h' \triangleright k \mid U\rangle\}\ .$$

Notice that $\mathcal{S}$ preserves destinations and lost locations, and that $\mathcal{S}^\star$ relates two nets if and only if one of them is obtained from the other by an arbitrary rearrangement of its forwarder trees. Our goal is to prove that $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor})$ is a candidate relation for Technique 4.11. This will allow us to prove that both $\mathcal{S}$ and $\xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor}$ are contained in bisimilarity. The two following lemmas establish progressively that $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor\mathrm{o}\rfloor})$ is a "simulation up to transitivity".

Lemma 4.15. *If* $U \xrightarrow{\tau}_{\lfloor o \rfloor} V$ *and* $U \xrightarrow{\mu}_{\lfloor o \rfloor} U'$*, then* $U' \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} \mathcal{S}^{\star} \overset{\widehat{\tau}}{\Leftarrow}_{\lfloor o \rfloor} \overset{\widehat{\mu}}{\leftarrow}_{\lfloor o \rfloor} V$*.*

Proof. The case $\mu = \tau$ is given by the local confluence of $\xrightarrow{\tau}_{\lfloor \mathrm{o} \rfloor}$ (Lemma 4.13); otherwise, it holds that $U' \xrightarrow{\tau}_{\lfloor \mathrm{o} \rfloor} \overset{\mu}{\leftarrow}_{\lfloor \mathrm{o} \rfloor} V$, except in the following case:

$$U = W \mid h\langle m \rangle_{\widetilde{h}} \mid h[n] \quad \xrightarrow{\tau}_{\lfloor \mathrm{o} \rfloor} \quad W \mid h[m;n] \mid \widetilde{h}\langle \triangleright h \rangle = V \qquad [\text{DST}_{\mathrm{O}}]$$
$$U \xrightarrow{h \triangleright k}_{\lfloor \mathrm{o} \rfloor} W \mid h\langle m \rangle_{\widetilde{h}} \mid h \triangleright k \mid k\langle n \rangle = U' \qquad [\text{MIG}_{\mathrm{O}}]$$

where we have

$$V \xrightarrow{h \triangleright k}_{\lfloor \mathrm{o} \rfloor} W \mid h \triangleright k \mid k\langle m;n \rangle \mid \widetilde{h}\langle \triangleright h \rangle = V' \,. \qquad [\text{MIG}_{\mathrm{O}}]$$

We reason by case analysis on the agent located at k:

- a localised process $k[n']$: by routing to k the messages exhibited in U' and V', we obtain:

  $$U' \overset{\tau}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} W' \mid (h, \widetilde{h}) \triangleright k \mid k[m;n;n']$$
  $$V' \overset{\tau}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} W' \mid \widetilde{h} \triangleright h \mid h \triangleright k \mid k[m;n;n']$$

  (the only message to route in V' is almost at its final destination, and the relocation message has already been sent to the blocked forwarders located at $\widetilde{h}$, so that the latter gets relocated under h instead of k).

  Finally, we relocate these forwarders with l applications of the swapping relation, l being the length of $\widetilde{h}$: $U' \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} \mathcal{S}^{l} \overset{\widehat{\tau}}{\Leftarrow}_{\lfloor \mathrm{o} \rfloor} V'$.
- A forwarder $k \triangleright k'$: like in the previous case, we first route the available messages to their destination, say k'':

  $$U' \overset{\tau}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} W' \mid (h, k, \widetilde{h}) \triangleright k'' \mid k''[m;n;n'] \;\; ,$$
  $$V' \overset{\tau}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} W' \mid \widetilde{h} \triangleright h \mid h \triangleright k \mid k \triangleright k'' \mid k''[m;n;n'] \;\; .$$

  Here we need an additional application of the swapping relation to relocate h to k'', before being able to relocate the forwarders at $\widetilde{h}$:

  $$U' \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} \mathcal{S}^{l+1} \overset{\widehat{\tau}}{\Leftarrow}_{\lfloor \mathrm{o} \rfloor} V' \;\; .$$

- A blocked forwarder $k \not\triangleright$. We reason like in the previous case by first routing the message that blocks this forwarder to its destination. ■

Lemma 4.16. *If* $U \;\mathcal{S}\; V$ *and* $U \xrightarrow{\mu}_{\lfloor o \rfloor} U'$ *then* $U' \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} \mathcal{S}^{\star} \overset{\widehat{\mu}}{\Leftarrow}_{\lfloor o \rfloor} V$*.*

Proof. It is immediate if $\mu \neq \tau$: we have $U' \;\mathcal{S} \overset{\mu}{\leftarrow}_{\lfloor \mathrm{o} \rfloor} V$. When $\mu = \tau$, the interesting cases are those where the silent transition $U \xrightarrow{\tau}_{\lfloor \mathrm{o} \rfloor} U'$ is the transmission of a message trough one of the two forwarders being swapped:

- $U = W \mid h\langle m\rangle_{\widetilde{h}} \mid h \triangleright h' \mid h' \triangleright k \;\xrightarrow{\tau}_{\lfloor o \rfloor}\; W \mid h\not\triangleright \mid h'\langle m\rangle_{h,\widetilde{h}} \mid h' \triangleright k = U'$.
  By routing the messages, we obtain:

$$\begin{aligned} U' &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k' \mid h' \triangleright k' \mid k'[m;n] = U'' \ , \\ V &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k' \mid h' \triangleright k \mid k'[m;n] = V' \ . \end{aligned}$$

  If $k = k'$, we are done: $U'' = V'$. Otherwise, there is a forwarder $k \triangleright k'$ in W', and we need one application of the swapping relation in order to relocate the forwarder located at h' in V'.
- $U = h \triangleright h' \mid h'\langle m\rangle_{\widetilde{h}} \mid h' \triangleright k \;\xrightarrow{\tau}_{\lfloor o \rfloor}\; h \triangleright h' \mid h'\not\triangleright \mid k\langle m\rangle_{h',\widetilde{h}} = U'$
  By routing the messages, we obtain:

$$\begin{aligned} U' &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright h' \mid h' \triangleright k' \mid k'[m;n] = U'' \ , \\ V &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k \mid h' \triangleright k' \mid k'[m;n] = V' \ . \end{aligned}$$

  If $k = k'$, we are done: $U'' = V'$. Otherwise, we have $W' = W'' \mid k \triangleright k'$, and we need two applications of the swapping relation in order to relocate the forwarder located at h in both nets:

$$\begin{aligned} U'' = {} & W'' \mid h \triangleright h' \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] \\ \mathcal{S} \ & W'' \mid h \triangleright k' \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] \\ \mathcal{S} \ & W'' \mid h \triangleright k \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] = V' \end{aligned}$$

This analysis also applies for the symmetric cases, where the silent transitions are played by the net with "flat" forwarders. ■

We now have to prove the termination property for Technique 4.11. Since $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^{+} \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} \subseteq (\mathcal{S}^{\star} \xrightarrow{\tau}_{\lfloor o \rfloor})^{+}$, the following lemma will suffice:

Lemma 4.17. *$\mathcal{S}^{\star} \xrightarrow{\tau}_{\lfloor o \rfloor}$ terminates.*

Proof. We call *size* of a net U the triple $s(U) = \langle n, r, l\rangle$, where n is the number of pending messages, r the number of relocation messages, and l the number of forwarders that are not blocked. These triples are ordered lexicographically. We check that $U \ \mathcal{S} \ V$ implies $s(U) = s(V)$, and that this size strictly decreases along silent transitions. ■

Proposition 4.18. *$\mathcal{S}$ and $\xrightarrow{\tau}_{\lfloor o \rfloor}$ are contained in weak bisimilarity.*

Proof. By applying Technique 4.11 to $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})$, we obtain that $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^{\star}$ is a simulation. Moreover, $\xleftarrow{\widehat{\tau}}_{\lfloor o \rfloor}$ is also a simulation (as is always the case for reversed silent transitions). By combining these two results we obtain that the symmetric relation $(\mathcal{S} \cup \xleftrightarrow{\tau}_{\lfloor o \rfloor})^{\star} = ((\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^{\star} \cup \xleftarrow{\widehat{\tau}}_{\lfloor o \rfloor})^{\star}$ is a simulation, and hence a bisimulation. We finally have $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor}) \subseteq (\mathcal{S} \cup \xleftrightarrow{\tau}_{\lfloor o \rfloor})^{\star} \subseteq \approx$. ■

Notice that Technique 4.11 plays a crucial role in the proof of the above proposition: it allows us to focus on the "local" relations $\mathcal{S}$ and $\xrightarrow{\tau}_{\lfloor o \rfloor}$, which relate nets that differ only slightly, so that we can reason about their small syntactical differences. In contrast, if we had to prove the previous proposition directly, we would have to show that $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^\star$ is a simulation; which is really hard as this relation relates completely different nets. Moreover, the technique avoids us to extend explicitly the swapping relation to situations where the involved forwarders may be blocked: these cases are captured implicitly by using silent transitions.

Also remark that the situation is reminiscent of that which appeared in the proof of Lemma 3.12 in the simple model: the proof has to handle jointly the silent transitions and the swapping relation; we need to reason up to $\mathcal{S}$ for silent transitions, and up to silent transitions for $\mathcal{S}$.

4.2.3 Relating simple nets to optimised nets

Lemmas 4.13 and 4.17 ensure that $\xrightarrow{\tau}_{\lfloor o \rfloor}$ defines a unique normal form for any clean optimised net U (termination of $\mathcal{S}^\star \xrightarrow{\tau}_{\lfloor o \rfloor}$ entails termination of $\xrightarrow{\tau}_{\lfloor o \rfloor}$); we denote by $U_{\downarrow o}$ this normal form. Notice that $U_{\downarrow o}$ is always a reference net: it does not contain any blocked forwarder, relocation message, nor pending, annotated messages.

The normalisation of a clean simple net by $\xrightarrow{\tau}_{\lfloor s \rfloor}$ and $\xrightarrow{\tau}_{\lfloor o \rfloor}$ does not necessarily lead to the same net: $U_\downarrow \neq U_{\downarrow o}$. However, these nets differ only by some rearrangement of their forwarder trees, they are related by $\mathcal{S}^\star$:

Lemma 4.19. *For any clean simple net U, we have: $U_\downarrow \; \mathcal{S}^\star \; U_{\downarrow o}$.*

Proof. We proceed by well-founded induction on U, using the termination of the relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$:

- If U is a reference net, we have $U_\downarrow = U = U_{\downarrow o}$.
- If $U \xrightarrow{\tau}_{\lfloor s \rfloor} U'$, since U is clean and simple, we have

$$\begin{aligned} &U = V \mid h_0\langle m\rangle \mid \Pi_{i<l}\; h_i \rhd h_{i+1} \mid h_l[n] \\ &U \xrightarrow{\tau}_{\lfloor s \rfloor} U' \overset{\tau}{\Rightarrow}_{\lfloor s \rfloor} V \mid \Pi_{i<l}\; h_i \rhd h_{i+1} \mid h_l[m;n] = U_1 && [\text{FWD}_\text{S},\text{DST}_\text{S}] \\ &U \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} V \mid \Pi_{i<l}\; h_i \rhd h_l \mid h_l[m;n] = U_2 && \text{(Lemma 4.12)} \end{aligned}$$

  We check that $U_1 \; \mathcal{S}^l \; U_2$, and we have $U_1 \overset{\hat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} U_{1\downarrow o}$ so that from Prop. 4.18, $U_2 \overset{\hat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} U_2'$ with $U_{1\downarrow o}(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^\star U_2'$.
  Furthermore, since $\mathcal{S}$ preserves normal forms, $U_{1\downarrow o} \; \mathcal{S}^\star \; U_2'$, and $U_2' = U_{2\downarrow o}$.
  Finally, by induction, $U_{1\downarrow} \; \mathcal{S}^\star \; U_{1\downarrow o}$ and $U_\downarrow = U_{1\downarrow} \; \mathcal{S}^\star \; U_{1\downarrow o} \; \mathcal{S}^\star \; U_{2\downarrow o} = U_{\downarrow o}$. ■

It follows that in the optimised model, any simple net is bisimilar to its normal form w.r.t. $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ (this correspond to Cor. 3.13 in the simple model):

Corollary 4.20. *For any clean simple net U, we have:*

$$\langle U, \rightarrow_{\lfloor o \rfloor} \rangle \approx \langle U_{\downarrow o}, \rightarrow_{\lfloor o \rfloor} \rangle \approx \langle U_{\downarrow}, \rightarrow_{\lfloor o \rfloor} \rangle \ .$$

Proof. By Lemma 4.19, $U \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor \mathrm{o} \rfloor} U_{\downarrow \mathrm{o}} \ \mathcal{S}^{\star} \ U_{\downarrow}$; we conclude with Prop. 4.18. ■

Since Corollary 4.20 does not give an expansion result, we cannot use the standard bisimulation up to expansion technique to give the final proof of correctness. Instead, we use the following restricted form of the "bisimulation up to bisimilarity" technique (in its unrestricted form, this technique is not correct [30]):

Technique 4.21 (Bisimulation up to Bisimilarity on Visible Actions)**.**
Let $\mathcal{R}$ be a relation such that:

- $\xleftarrow{\tau} \mathcal{R} \subseteq \mathcal{R} \overset{\widehat{\tau}}{\Leftarrow}$ and $\mathcal{R} \xrightarrow{\tau} \subseteq \overset{\widehat{\tau}}{\Rightarrow} \mathcal{R}$; and
- $\xleftarrow{a} \mathcal{R} \subseteq \approx\mathcal{R}\approx \overset{a}{\Leftarrow}$ and $\mathcal{R} \xrightarrow{a} \subseteq \overset{a}{\Rightarrow} \approx\mathcal{R}\approx$ for any visible action $a \in \mathcal{L} \setminus \{\tau\}$.

Then $\approx\mathcal{R}\approx$ is a bisimulation, and $\mathcal{R} \subseteq \approx$.

Using this technique, we can consider only reference nets, that do not contain any pending messages, and normalise them whenever they do a visible action that brings some pending messages. Since reference nets do not give rise to silent transitions, even when considered in the simple or optimised models, the restriction of the previous technique to visible challenges is not problematic.

Lemma 4.22. *For any reference net U, we have:*

$$\langle U, \rightarrow_{\lfloor o \rfloor} \rangle \approx \langle U, \rightarrow_{\lfloor s \rfloor} \rangle \ .$$

Proof. We apply Technique 4.21 to the following relation:

$$\mathcal{R} \triangleq \left\{ \left\langle \langle U, \rightarrow_{\lfloor \mathrm{o} \rfloor} \rangle, \ \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \right\rangle \mid U \in \mathcal{U}_{\mathrm{r}} \right\} \ .$$

For any reference net U, since U does not contain any pending message, we have $U \xrightarrow{\mu}_{\lfloor \mathrm{o} \rfloor} V$ iff $U \xrightarrow{\mu}_{\lfloor \mathrm{s} \rfloor} V$. However, while V is a clean simple net, it is not necessarily a reference net, so that we do not have $\langle V, \rightarrow_{\lfloor \mathrm{o} \rfloor} \rangle \ \mathcal{R} \ \langle V, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$. Nevertheless, $V_{\downarrow}$ is a reference net, and by Corollaries 4.20 and 3.13, we have:

$$\langle V, \rightarrow_{\lfloor \mathrm{o} \rfloor} \rangle \approx \langle V_{\downarrow}, \rightarrow_{\lfloor \mathrm{o} \rfloor} \rangle \ \mathcal{R} \ \langle V_{\downarrow}, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \precsim \langle V, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \ .$$

We use bisimilarity to rewrite the left-hand-side process; this is allowed by Technique 4.21: we are necessarily in a visible challenge. ■

By gathering the previous results, we can finally prove the correctness of the implementation w.r.t. the specification:

Theorem 4.23. *For any optimised net U, we have:*

$$\langle U, \to_o \rangle \approx \langle \lfloor U \rfloor_{\downarrow o}, \to_r \rangle \ .$$

Proof. $\lfloor U \rfloor_{\downarrow \mathrm{o}}$ is a reference net, therefore, we have:

$$\begin{aligned} \langle U, \to_{\mathrm{o}} \rangle &\succsim \langle \lfloor U \rfloor, \to_{\lfloor \mathrm{o} \rfloor} \rangle && \text{(Proposition 4.10)} \\ &\approx \langle \lfloor U \rfloor_{\downarrow \mathrm{o}}, \to_{\lfloor \mathrm{o} \rfloor} \rangle && \text{(Corollary 4.20)} \\ &\approx \langle \lfloor U \rfloor_{\downarrow \mathrm{o}}, \to_{\lfloor \mathrm{s} \rfloor} \rangle && \text{(Lemma 4.22)} \\ &\succsim \langle \lfloor U \rfloor_{\downarrow \mathrm{o}}, \to_{\mathrm{r}} \rangle \ . && \text{(Theorem 3.16)} \end{aligned}$$

■

5 Concluding Remarks

We conclude by giving an overview of the previous proof, discussing related work, and giving directions for future work.

5.1 Structure of the proof

The main steps of the correctness proof are summed up below; we now compare the up-to techniques that were used for each step mentioned in this picture.

$$\begin{array}{ccccc} & & \text{simple} & & \text{optimised} \\ & & \rotatebox{90}{$\succsim$} & & \rotatebox{90}{$\succsim$} \\ \text{reference} & \precsim & \text{clean, simple} & \approx & \text{clean, optimised} \end{array}$$

While a direct expansion proof has been sufficient to remove lost messages from simple nets, we had to use an “expansion up to transitivity” technique in order to do so in the optimised model. In both cases, we used the “expansion up to expansion” technique in order to deduce from this property of the simple and optimised models that we could work in the clean models, where lost messages are systematically removed after each transition.

Then we had to show that silent transitions are contained in weak bisimilarity in the two clean models. While we could use the "expansion up to transitivity" in the simple case (silent transitions of this model are actually contained in expansion), we had to use a constrained form of "weak bisimulation up to transitivity" in the optimised case (Technique 4.11, from [26], that requires a termination hypothesis). Since optimised silent transitions involve some rearrangement of forwarder trees, we had to introduce the "swapping" relation ($\mathcal{S}$), and show that the latter is contained in weak bisimilarity. Thanks to the up to transitivity technique, we were able to consider this local relation, rather than its reflexive transitive closure ($\mathcal{S}^\star$), which is more complex.

Finally, we gave an "expansion up to expansion" proof in order to relate the simple clean model to the reference model, and we used the "bisimulation up to weak bisimilarity" technique, restricted to visible challenges, in order to relate optimised and clean nets to simple and clean ones.

In both cases, the various up-to techniques we used made it possible to reason about local relations only.

5.2 Related work

We split this section into three parts: related work on distributed abstract machines, other uses of forwarders, and other uses of up-to techniques.

5.2.1 Other distributed abstract machines or frameworks

As hinted in the introduction, several distributed implementations have been defined recently, for various kinds of calculi featuring mobility. Among these works, the closest to our framework is certainly the study of *location independence* for Nomadic Pict, by Unyapoth and Sewell [35]: in a language with mobility, we would like to be able to communicate with an agent without having to track its movements. This is exactly what we provide here: if we know that an agent is hosted at location h at some point, messages sent to h will always reach that agent, independently from its migrations. The differences with [35] are the following: (1) they consider *physical* locations, while we work at the slightly more abstract level of *logical* locations; as a side effect, their *unlocated labels* – that closely correspond to our *process labels* – do not include a label for location creation: this does not make sense for physical sites; (2) they achieve location independence by a centralised forwarding algorithm: a dedicated server is used to track down the movements of agents and to forward location independent messages accordingly. While we presented only asynchronous implementations here, we could define such a centralised, synchronous model in our setting, and prove its correctness. Conversely, their centralised infrastructure could be replaced by one of our asynchronous models (following their idea that the choice of the adequate infrastructure should be application-specific). Notice that Distributed Join [11] is also a calculus that provides location independent communications.

The framework presented here comes from a distributed abstract machine for Safe Ambients [31]. There are at least three other implementations of ambient calculi: the *Ambit* machine of Cardelli [4], which is not distributed; the *AtJ* machine of Fournet, Lévy and Schmitt [12], which is distributed and asynchronous, and uses JoCaml[9] as a target language – we discuss this machine below; and a machine from Phillips, Yoshida and Eisenbach [25], that implements a variant of Boxed Ambients [3] (called Channel Ambient) where the *open* primitive is replaced by additional communication capabilities. The latter machine is distributed but synchronous: it uses locks to implement synchronisation patterns. We refer to [31,17] for a more detailed comparison of those distributed abstract machines for ambient calculi; notice that ambient calculi do not provide directly location independent communications (communications are local); with Channel Ambients for example, such communication patterns have to be recovered by defining a protocol in the language itself (cf. the "ambient tracker" application [25]).

5.2.2 Using forwarders

Forwarders have been widely used in the literature: they play an important role in low-level protocols like TCP/IP, or in the implementation of distributed programming languages (e.g., "stub-scion pairs" [33] used for implementing distributed references). Le Fessant used such a mechanism [20] in its implementation of Distributed Join [11] (JoCaml [9]); although strong similarities with our work exist (contraction of forwarder chains for example), the mechanisms involved are technically different: in order to achieve garbage collection, Le Fessant's forwarders are not "multiplexed": only one forwarder can point to another one, so that there are only chains of forwarders, not trees.

For their implementation of Mobile Ambients (AtJ [12]), Fournet, Lévy and Schmitt rely on the migration primitives provided by the JoCaml language. However, they still need forwarders in order to implement the *open* primitive. This is also for this particular primitive that forwarders are used in the abstract machine for Safe Ambients, by Sangiorgi and Valente (the PAN [31], that initially motivated the analysis of the framework presented here). This primitive being absent from the Channel Ambients calculus, the machine by Phillips et al. [25] does not require such a mechanism. While the PAN can be implemented in our framework (obviously, by design), it is unclear whether the optimisation introduced here would be useful in the case of the AtJ machine, where the role of forwarders is not essential.

Gardner, Laneve and Wischik also used forwarders, first in order to implement the calculus of "explicit fusions" [13], and then to implement the asynchronous π-calculus [14]. In the fusion machine, forwarders are created in order to implement the substitutions that appear along with reactions (the "explicit fusions"); the *union*-part of Tarjan's union-find algorithm [34] is used in order to optimise these distributed substitutions, the *find*-part of that algorithm (contraction of chains) is not implemented however. It seems straightforward to implement this machine using our framework, which would give a further optimisation. Although the second work by Gardner et al. [14] actually comes from the work on the fusion machine, the forwarders they use there have a different behaviour: first of all, they are *linear*, in the sense that they are intended to transmit only one message; therefore, there is no point in trying to contract forwarder chains. Secondly, rather than small processes executed at some source location in order to redirect a potential message to a target location, they should be thought of as a message, sent to the source location in order to inform that location that there is a "listener" at the target location.

5.2.3 Using up-to techniques

Up-to techniques based on the expansion preorder [30] are certainly the most popular ones: Sangiorgi combined them with the "up to context" technique [29], in order to reason about the π-calculus [32] (notably for the encoding of the λ-calculus). Notice that while we used techniques based on expansion, reasonning "up to contexts" was not required in the present work. Merro and Nardelli [22] extended the "weak bisimulation up to expansion" technique to Mobile Ambients [5]: they used it in order to validate some laws of this calculus. Unyapoth and Sewell also used the "expansion up to expansion" technique in order to reason about their infrastructure for Nomadic Pict [35].

Jeffrey and Rathke defined the "up to β" technique [19] (β are *deterministic* transitions), that they use quite intensively in order to prove the completeness of a bisimulation proof method for Concurrent ML [28]. This technique, which is actually a very special case of "up to expansion" (β transitions form an expansion relation), has also been used by Hennessy et al. [7,18] in the setting of Distributed Pi [16]. Fournet also defines such an "up to deterministic reductions" technique [8], that he uses to reason about the Join calculus [10], and to prove the correctness of the AtJ machine for Mobile Ambient [12]. Unlike "up to β" these techniques go slightly beyond expansion; they rely on the ideas of *decreasing diagrams for confluence*, by van Oostrom [36].

Although these "up to deterministic transitions" techniques correspond to our way of reasoning "up to silent transitions" when we prove the correctness of both implementations, the situation we describe in this article is more demanding, as far as up to techniques are concerned: in the simple model, we need an "up to transitivity" technique in order to prove that silent transitions are contained in expansion; moreover, in the optimised model, where these transitions are contained in weak bisimilarity only, the technique based on termination guarantees is required.

5.3 Directions for future work

In order to reason about clean models, our strategy was to refine the initial model by using a function to "normalise" the reducts after each transition. This method seems to be quite generic, and it would be nice to obtain general results about this kind of refinements: what are the requirements in order for an LTS refined in such a way to remain equivalent to the initial one? In the present work, we used expansion results between nets and their cleaned forms (both in the simple and optimised models); as expansion may not always hold however, it would be interesting to understand how the use of the "up to expansion" technique could be replaced by more elaborate techniques, when reasoning about such refined LTSs.

As hinted in section 3.2, an LTS whose silent transitions are contained in weak bisimilarity is commonly called τ-inert [23]. This property, which we proved to be satisfied by simple and optimised models, is generally quite useful: it states that the internal behaviour of a system does not disturb its external behaviour. Groote and Sellink [15] have given some "up to transitivity" techniques to prove the τ-inertness of a system whose silent transitions happen to be terminating. Their results are weaker than ours (Technique 4.11 [26]), and they cannot be used in our setting: basically they do not allow one to reason simultaneously about silent transitions and rearrangement of forwarder trees. We plan to extend the work of Groote and Sellink, by giving a general account of how the results from [26] can be used to obtain more powerful techniques for proving τ-inertness, under some termination hypotheses.

Acknowledgements

I would like to thank Daniel Hirschkoff, whose great help and comments have been essential during the preparation of this paper.

References

[1] S. Arun-Kumar, M. Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29(9), 1992.

[2] P. Bidinger, J.-B. Stefani. The Kell Calculus: Operational Semantics and Type System. In *Proc. of FMOODS '03*, vol. 2884 of *LNCS*. Springer Verlag, 2003.

[3] M. Bugliesi, G. Castagna, S. Crafa. Boxed Ambients. In *Proc. TACS '01*, vol. 2215 of *LNCS*. Springer Verlag, 2001.

[4] L. Cardelli. Ambit, 1997. `http://www.luca.demon.co.uk/Ambit/Ambit.html`.

[5] L. Cardelli, A. Gordon. Mobile Ambients. In *Proc. FOSSACS '98*, vol. 1378 of *LNCS*. Springer Verlag, 1998.

[6] G. Castagna, F. Z. Nardelli. The Seal Calculus Revisited. In *Proc. of FSTTCS '02*, vol. 2556 of *LNCS*. Springer Verlag, 2002.

[7] A. Ciaffaglione, M. Hennessy, J. Rathke. Proof methodologies for behavioural equivalence in DPI. In *Proc. FORTE '05*, vol. 3731 of *LNCS*. Springer Verlag, 2005.

[8] C. Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, 1998.

[9] C. Fournet, F. L. Fessant, L. Maranget, A. Schmitt. JoCaml: A Language for Concurrent Distributed and Mobile Programming. In *Proc. of Advanced Functional Programming 2002*, vol. 2638 of *LNCS*. Springer Verlag, 2002.

[10] C. Fournet, G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[11] C. Fournet, G. Gonthier. The join calculus: A language for distributed mobile programming. In *Proc. APPSEM*, LNCS. Springer Verlag, 2000.

[12] C. Fournet, J. Lévy, A. Schmitt. An Asynchronous, Distributed Implementation of Mobile Ambients. In *Proc. IFIP TCS'00*, vol. 1872 of *LNCS*. Springer Verlag, 2000.

[13] P. Gardner, C. Laneve, L. Wischik. The fusion machine. In *Proc. CONCUR '02*, vol. 2421 of *LNCS*. Springer Verlag, 2002.

[14] P. Gardner, C. Laneve, L. Wischik. Linear forwarders. *Information and Computation*, 205(10), 2007.

[15] J. F. Groote, M. P. A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1-2), 1996.

[16] M. Hennessy. *A Distributed Pi-Calculus.* Cambridge University Press, 2007.

[17] D. Hirschkoff, D. Pous, D. Sangiorgi. A Correct Abstract Machine for Safe Ambients. In *Proc. COORD '05*, vol. 3454 of *LNCS*. Springer Verlag, 2005.

[18] S. Hym, M. Hennessy. Adding recursion to Dpi. *Theoretical Computer Science*, 373(3), 2007.

[19] A. Jeffrey, J. Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theoretical Computer Science*, 323(1-3), 2004.

[20] F. Le Fessant. *JoCaml: conception et implémentation d'un langage à agents mobiles*. PhD thesis, École Polytechnique, 2001.

[21] F. Levi, D. Sangiorgi. Mobile Safe Ambients. *ACM Trans. on Progr. Lang. and Sys.*, 25(1), 2003. ACM Press.

[22] M. Merro, F. Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. 30th ICALP*, vol. 2719 of *LNCS*. Springer Verlag, 2003.

[23] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[24] R. D. Nicola, G. Ferrari, R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Trans. Soft. Eng.*, 24(5), 1998.

[25] A. Phillips, N. Yoshida, S. Eisenbach. A Distributed Abstract Machine for Boxed Ambient Calculi. In *ESOP '04*, LNCS. Springer Verlag, 2004.

[26] D. Pous. Up-to Techniques for Weak Bisimulation. In *Proc. 32nd ICALP*, vol. 3580 of *LNCS*. Springer Verlag, 2005.

[27] D. Pous. On bisimulation proofs for the analysis of distributed abstract machines. In *Proc. TGC '06*, vol. 4661 of *LNCS*. Springer Verlag, 2006.

[28] J. H. Reppy. CML: A higher-order concurrent language. In *Proc. PLDI*. ACM, 1991.

[29] D. Sangiorgi. On the Bisimulation Proof Method. *Journal of Mathematical Structures in Computer Science*, 8, 1998.

[30] D. Sangiorgi, R. Milner. The problem of "Weak Bisimulation up to". In *Proc. 3rd CONCUR*, vol. 630 of *LNCS*. Springer Verlag, 1992.

[31] D. Sangiorgi, A. Valente. A Distributed Abstract Machine for Safe Ambients. In *Proc. 28th ICALP*, vol. 2076 of *LNCS*. Springer Verlag, 2001.

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

[33] M. Shapiro, P. Dickman, D. Plainfossé. Robust, distributed references and acyclic garbage collection. In *Proc. PODC '92*. ACM Press, 1992.

[34] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of ACM*, 22(2), 1975.

[35] A. Unyapoth, P. Sewell. Nomadic pict: Correct Communication Infrastructure for Mobile Computation. In *Proc. 28th POPL*. ACM Press, 2001.

[36] V. van Oostrom. Confluence by Decreasing Diagrams. *Theoretical Computer Science*, 126(2), 1994.