Distributed Directory Service and Message Routing for Mobile Agents

Luc Moreau
Department of Electronics and Computer Science
University of Southampton, UK
`L.Moreau@ecs.soton.ac.uk`

ECSTR M99/3

November 22, 1999

Abstract

Research about networks and agents has identified the need for a layer that provides a uniform protocol to communicate with fixed and mobile agents. In order to preserve the compatibility with existing infrastructures, proposed solutions have involved a "home agent", which forwards messages to a mobile entity. The mechanism of a home agent puts a burden on the infrastructure, which may hamper the scalability of the approach, in particular, in massively distributed systems, such as the amorphous computer or the ubiquitous/pervasive computing environment. Free from any compatibility constraint, we have designed an algorithm to route messages to mobile agents that does not require any fixed location. The algorithm has two different facets: a *distributed directory service* that maintains distributed information about the location of a mobile agent, and a *message router* that uses the directory service to deliver messages to a mobile agent. Two properties of the algorithm were established. The safety property ensures that messages are delivered to the agent they were aimed at, whereas the liveness property guarantees that messages eventually get delivered. A mechanical proof of the properties was carried out using the proof assistant Coq.

1 Introduction

Mobile hardware, ranging from cellular phones to personal digital assistants (PDAs) are an important trend in the consumer electronics market. We can foresee the time when all their functionalities will be merged into a single device communicating via a form of Internet. *Mobile computing* is generally the term used to denote this area of research. Cardelli [7] convincingly argues that *mobile computation*, denoting mobile software such as mobile agents [14, 17, 18, 27, 32], share similar problems as mobile computing. Security, resource discovery, and communication issues are identical for a software running on a laptop that was just connected to a foreign network or for a mobile agent that has been allowed to penetrate a new domain.

There are numerous research topics related to mobility [6], including network communications [9], mobile code security [26], active routing [13] and firewalls modelling [8]. Several calculi have been devised to reason about and to study those issues, including ambients [8], seal-calculus [33], spi-calculus [1] or join calculus [11].

In this paper, we focus on communications between mobile agents; in order to introduce the problem we are addressing, we present an analogy with phones. Before the advent of mobile phone, tedious procedures were required to contact mobile users. We had to call their secretary or their pager, leave a message, and wait for their callback; or, if we knew what their location was, we could attempt to call them there, possibly repeating that procedure if we were told what their next destination was. Whatever approach we adopted, it was more complex than straight dialing to a fixed location. With mobile phone technology, phone companies now provide a single protocol[1] for communicating with fixed or mobile users.

From a programming viewpoint, it is also convenient to program communications with mobile agents similarly as with fixed locations. Various transport layers have been proposed and implemented; they belong to different software layers and therefore have different purposes and provide different services. For instance, the version 6 of the IP protocol (IPv6) supports mobile IP addresses [9, 15]; the agent programming language APRIL [18], the asynchronous InterAgent Communications Model (ICM) [19] and the FIPA proposal for mobile agents [10] provide uniform addressing of mobile agents.

Using communications between fixed locations, those layers route messages in order to provide communications between mobile entities. The techniques adopted vary substantially. In order to preserve compatibility with the IP protocol, IPv6 associates each mobile agent with a *home agent*. Fixed hosts using IP, which are unaware of mobility, communicate with the home agent that tunnels messages to the mobile agent. IPv6 does not provide reliable communications, but relies on a transport layer for that matter. APRIL and ICM introduce agent names that contain routing information, and assume the presence of a fixed home agent; their store and forward architecture provides reliable, though not necessarily in-order, message delivery.

Even though we understand the motivation that lead to these designs, a home base agent is a fixed resource that puts a burden on the global infrastructure, and therefore

[1]Conventions in phone numbers indicate if a number refers to a mobile or a fixed station; however, routing is not made visible to the user.

may prevent the scalability of the approach. In particular, in a ubiquitous/pervasive computing environment [34, 2], let us consider mobile devices that establish a communication when their owners meet in a room: the solution requiring them to communicate via their host agent, possibly on the other side of the planet, does not appear as the most natural solution, because a local communication medium could be used instead. In addition, the assumption behind an agent home base is that it can be reached from any other node in the network: such a requirement is obviously not valid in the presence of firewalls[2].

In this paper, we investigate an algorithm for transporting messages between mobile agents. Our undertaking is based on the following assumptions: *(i)* We set ourselves free to design an algorithm without necessarily preserving the compatibility with an existing infrastructure. *(ii)* We wish to design a distributed algorithm without any fixed or centralised control. *(iii)* Our design takes place at the *application level*, where we wish to build a mobile agent system: therefore message delivery must be reliable. *(iv)* The correctness of the algorithm has to be established so that further services using that layer may themselves be proven correct; the long term goal of this effort is the design of a secure mobile agent system.

The proposed algorithm has two distinct facets: a *distributed directory service* that maintains distributed information about the location of a mobile agent, and a *message router* that uses the directory service to deliver messages to mobile agents. In order to prove the correctness of the algorithm, two properties are established: the *safety* of the distributed directory service ensures that it correctly tracks the mobile agent's location; its *liveness* guarantees that agent location information eventually gets propagated. Similar safety and liveness properties are established for the message router. The proofs of these properties have been carried out using the proof assistant Coq [4]; complete proofs may be downloaded from [22].

This paper is organised as follows. In Section 2, we informally present the algorithms for the distributed directory service and the message router. (A preliminary version of these algorithms was previously sketched [21], but no attempt was made to formalise them at that time.) Then, each algorithm is formally defined in Sections 3 and 4: in both cases, the formalisation takes the form of an abstract machine modelling an asynchronous distributed system. The correctness of the algorithms is established in Section 5. The algorithms are then discussed and compared with related work in Section 6.

2 Informal Presentation of the Algorithm

Let us define some terminology, before intuitively introducing the algorithm. We assume that a finite set of *sites* take part to a computation; sites are uniquely identified fixed machines that may execute the code of an agent. *Agents* are mobile and have the ability to migrate from sites to sites. We do not address the problem of security here: in real life, a site may run some security checks [26] before deciding to accept or reject the visit of a mobile agent. Sites are able to *exchange messages*. For the time being, we also

[2]Note that workarounds were introduced for ICM [19] to deal with this issue.

assume that communications can take place between any pair of sites; we will discuss the presence of firewalls in Section 6.

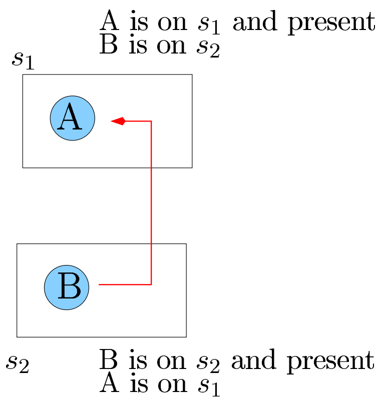

Figure 1: Routing of a Message from B to A

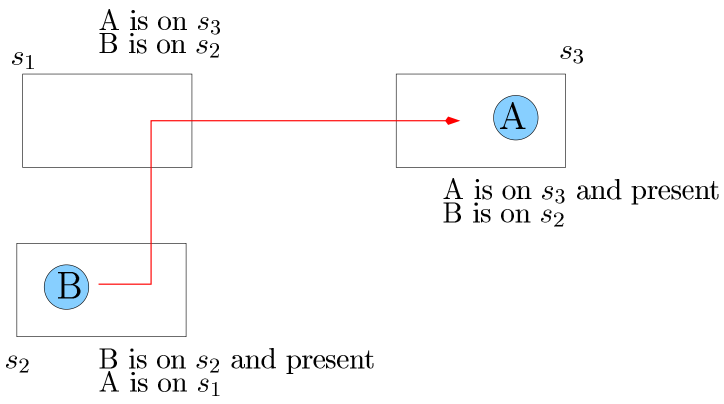

Figure 2: Forwarding of a Message by s_1

At first, let us consider that each site maintains a local table containing locations where agents are thought to be. In Figure 1, we have two agents A and B, and three sites s_1, s_2, s_3. Site s_1 knows that A is local and B is on s_2, whereas site s_2 knows that B is local and A is on s_1. Our presentation will show how each site has reached such a local knowledge. The information that is distributed between sites s_2 and s_1 is sufficient to route messages from B to A. If B wishes to send a message to A, it requests its site s_2 to deliver the message on its behalf. As s_2 knows that the agent A is on s_1, the message may be sent from s_2 to s_1, which in turn can deliver the message to A, known to be local.

Figure 2 displays the situation where agent A has migrated to s_3. Local knowledge has been updated on s_1 and s_3, which are now both aware that the agent A is on s_3. Again, our presentation will explain how these tables are modified, but we can already observe that the knowledge of s_2 is unchanged. Therefore, when s_2 is requested to deliver a message to the agent A, it still sends the message to s_1, which in turn forwards it to

s_3; finally, s_3 is able to deliver the message to the locally present agent A.

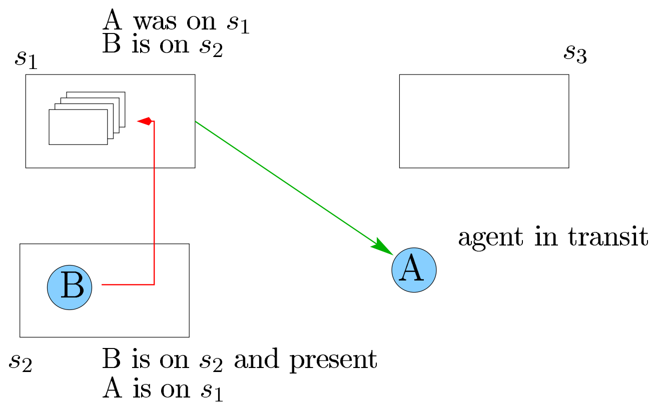

Figure 3: Non Atomic Agent Migration

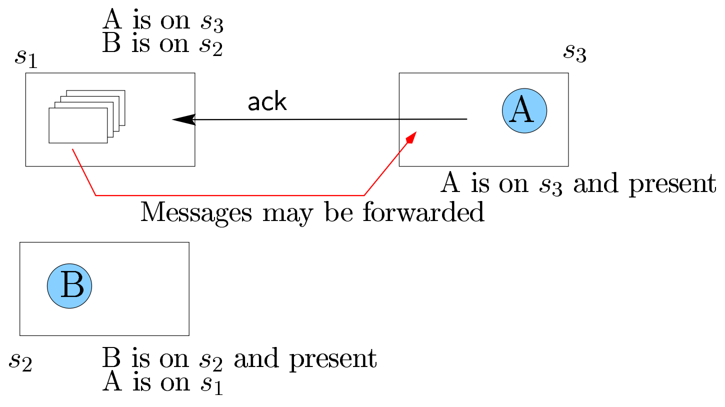
A is on s_3
B is on s_2
s_1
s_3
ack
A
A is on s_3 and present
Messages may be forwarded
B
s_2
B is on s_2 and present
A is on s_1

Figure 4: Acknowledgement Following an Agent Migration

In practice, agent migration is not atomic: some finite time is required in order to transform the state of Figure 1 into the state of Figure 2. For instance, a mobile agent may be transported in a laptop: hours or even days may pass between the disconnection and the reconnection of the laptop. Figure 3 represents the scenario where the agent A has left site s_1 and is in transit. Local knowledge on s_1 was updated to indicate that A was present on s_1, but its new location is still unknown. When an agent is in transit, it is disconnected from a fixed host, and is not able to receive any message. Therefore, in order to provide reliable communications, messages aimed at A have to be enqueued on s_1, until the new agent's position is known.

As an agent arrives at its destination, its previous site must be informed of its new location. Such an action is represented in Figure 4 by an ack message which acknowledges the safe arrival of the agent. When a site s_1 receives an ack message related to an agent A from a site s_3, it can update its local knowledge about A and forward all accumulated messages to the new location.

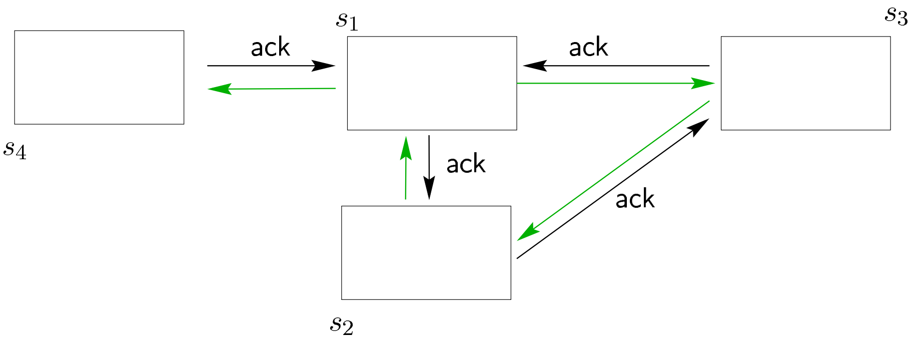

Figure 5: Conflicting Acknowledgement Messages

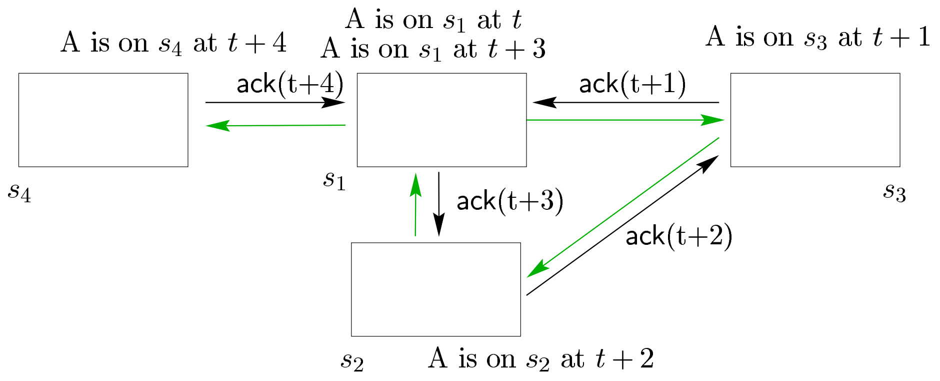

Figure 6: Timestamps

So far, we have described an algorithm by which an agent location is propagated backwards as the agent migrates, but *unsafe updating* may take place when an agent migrates to a site that it has already visited. Figure 5 describes such a scenario: starting from s_1, the agent successively migrates to s_3, s_2, back to s_1 and then to s_4. For every agent migration represented by a gray arrow, there is an acknowledgement message in the opposite direction (black arrow). There is potentially a race condition between two acknowledgement messages in transit to s_1 from s_3 and s_4. As a result, old information may overwrite more recent information, because the ack message from s_3 could be processed by s_1 after the ack messsage from s_4. This may lead to the following unsafe situation: s_1, s_2, s_3 have a local knowledge indicating that the agent is on s_3, s_2, s_1, respectively. Consequently, any remaining message in transit between those sites will indefinitely be forwarded in a cycle, and will never reach the agent.

In order to avoid updating recent knowledge by older knowledge, we associate an agent with a timestamp, which we call *mobility counter*, which is incremented every time the agent changes location. In addition, each site maintains not only the location where an agent is believed to be, but also the timestamp it had at that moment. Furthermore, acknowledgement messages contain the mobility counter that the agent had when it reached its new location. When a site receives an acknowledgement message, it updates its local knowledge only if the message has a higher timestamp than its local knowledge. Figure 6 reconsiders the scenario of Figure 5, but making mobility counters explicit, with t its initial value on s_1. As illustrated, the two acknowledgement messages aimed at s_1 are respectively timestamped $t+1$ and $t+4$; if the message from s_3 is received after the message from s_2, the table will not be updated, which avoids the creation of a cycle.

The algorithm as presented has been proven safe; it is however inefficient as it leaves trails of forwarding pointers, which, in the worst case, make the cost of communication proportional to the number of agent migrations. It is essential to "short-cut" those chains of forwarding pointers in order to reduce the distributed state: this will avoid dependencies on visited sites, and will keep the cost of communications low.

We introduce a general mechanism by which any site may communicate its knowledge to any other site, at any moment. For this purpose, we use a new message `inform`, which contains an agent's location and the timestamp it had when it was at that location. A site that receives an inform message is allowed to update its table if the received information is more up-to-date than the one it had.

In this algorithm, we do not specify when inform messages must be sent, and which site they should be sent to. This is a policy that must be defined according to the actual distributed system where the algorithm is used. In a small distributed system with fast communications, broadcasting the inform message to all nodes may be a realistic solution, whereas it does not seem feasible in the Internet. Therefore, at this level, we consider a general solution, which we can prove to be correct; we leave the discussion of some policies to Section 6.

3 Distributed Directory Service

From a software engineering viewpoint, it is useful to separate the part of the algorithm that maintains the agent's location, from the one that deals with message forwarding. In this Section, we formalise the former one, which is a *distributed directory service*; the latter one, *the message router*, will be the object of the following section. Separating the two algorithms is beneficial, because there may be other algorithms than the message router that may reuse the distributed directory service.

Following previous work [23], we formalise the algorithm by an abstract machine, whose state space is displayed in Figure 7. The presentation very closely follows our encoding of the abstract machine in the proof assistant Coq [22]. The algorithm assumes that agents are referred to by their *name*; the role of the directory service is to map an agent name to its location. For the sake of modelling and proof simplicity, we formalise the algorithm for a single agent; it is straightforward to generalise it to the case of multiple agents[3].

A finite number of sites are involved in the abstract machine. The set of messages exchanged between sites is defined by an inductive type; its three constructors are named

[3]In the case of multiple agents, the algorithm requires unique names to be allocated to agents. Such names can easily be created by nodes, combining a unique address and a local naming scheme.

$$\begin{array}{rcll}
\mathcal{S} & = & \{s_0, s_1, \ldots, s_{n_s}\} & \text{(Set of Sites)} \\
\mathcal{M} & = & \mathsf{agent} : \mathcal{L} \to \mathcal{M} \ \mid \ \mathsf{ack} : \mathcal{L} \to \mathcal{M} & \text{(Messages)} \\
 & & \mid \ \mathsf{inform} : \mathcal{L} \times \mathcal{S} \to \mathcal{M} & \\
\mathcal{L} & = & \mathbf{Z} & \text{(Mobility Counters)} \\
\mathcal{K} & = & \mathcal{S} \times \mathcal{S} \to \mathit{Queue}(\mathcal{M}) & \text{(Message Queues)} \\
\mathcal{LT} & = & \mathcal{S} \to \mathcal{S} & \text{(Location Tables)} \\
\mathcal{PT} & = & \mathcal{S} \to \mathit{Bool} & \text{(Present Tables)} \\
\mathcal{MT} & = & \mathcal{S} \to \mathcal{L} & \text{(Mobility C. Tables)} \\
\mathcal{AT} & = & \mathcal{S} \to \mathcal{A} & \text{(Acknowledgement Tables)} \\
\mathcal{A} & = & \mathsf{negative} : \mathcal{A} \ \mid \ \mathsf{positive} : \mathcal{S} \times \mathcal{L} \to \mathcal{A} & \text{(Acknowledgement Info)} \\
\mathcal{C} & = & \mathcal{LT} \times \mathcal{PT} \times \mathcal{MT} \times \mathcal{AT} \times \mathcal{K} & \text{(Configurations)}
\end{array}$$

Characteristic variables:

$$s \in \mathcal{S}, \quad m \in \mathcal{M}, \quad k \in \mathcal{K}, c \in \mathcal{C}$$
$$\mathit{location\_T} \in \mathcal{LT}, \quad \mathit{present\_T} \in \mathcal{PT}, \quad \mathit{mob\_T} \in \mathcal{MT}, \quad \mathit{ack\_T} \in \mathcal{AT}$$

Figure 7: State Space

according to the messages presented in Section 2, namely agent, ack, inform, with the message agent representing an agent in transit. Communication channels are represented by queues[4] of messages between pairs of sites. We use the following notations and operations on queues:

$q, q_1, \ldots$	:	denotes queues;
$\emptyset$	:	denotes the empty queue;
$first(q)$	:	head of a non empty-queue q;
$q \ \S \ \{m\}$	:	queue q after adding a message m at its tail;
$q_1 \ \S \ q_2$	:	queue obtained after concatenating q_1 and q_2.

Each site maintains a table called the *location table*, which records where the site believes the agent is located; the location table is represented as a function taking a site and returning a site.

When the agent migrates, there is a period of time during which it is in transit, and from the message router viewpoint, messages aimed at this agent must be stored away. For this purpose, we introduce an extra table, *the present table*, that indicates whether the agent is in transit. In our semantics, we shall preserve the following meaning for $\mathit{location\_T}$ and $\mathit{present\_T}$:

1. If $\mathit{location\_T}(s) = s'$, with $s \neq s'$ then $\neg \mathit{present\_T}(s)$.

2. If $\mathit{location\_T}(s) = s$ and $\mathit{present\_T}(s)$, then the agent is on site s.

[4]Our formalisation follows closely the one for a distributed reference counting algorithm [23], which requires fifo communication channels. We shall see later that the restriction on fifo delivery may be lifted.

3. If $location\_T(s) = s$ and $\neg present\_T(s)$, then the agent was on site s, but is now in transit; site s is not yet aware of its new position.

We always have the following implication: if $present\_T(s)$, then $location\_T(s) = s$.

The algorithm associates the agent with a counter, called *mobility counter*, which indicates the number of times the agent has migrated. A further table is introduced: the *mobility table*; it is used in conjunction with the location table. The latter maintains the location where the agent is thought to be, whereas the mobility table contains the mobility counter the agent had at the time.

Finally, when the agent reaches a new destination, an acknowledgement message has to be sent back to its previous location. It is convenient to decouple the agent's arrival from the acknowledgement sending, so that transitions that deal with incoming messages are different from those that generate new messages. Consequently, we introduce a further table, the *acknowledgement table*, which indicates if a site still has to acknowledge the arrival of an agent.

A configuration of the distributed system is defined as a tuple, composed of a location table, a present table, a mobility counter table, an acknowledgement table, and messages queues. Following this definition, we can state that we have modelled an asynchronous distributed system [16].

In the algorithm, we use some notations such as *post*, *receive* or table updates, which give an imperative look to the algorithm; their definitions appear in Figure 8. The directory service algorithm is itself encoded by transitions of the abstract machine, as displayed in Figure 9. Transitions are defined as inductive types, whose constructors are migrate_agent, receive_agent, ack_agent, receive_ack and receive_ack2. Three additional transitions are defined in Figure 10 to reduce chains of forwarding pointers: informt, receive_inform, receive_inform2.

Given a configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$,

- $mob\_T(s) := V$ denotes $c = \langle location\_T, present\_T, mob\_T', ack\_T, k\rangle$, such that $mob\_T'(s) = V$ and $mob\_T'(s') = mob\_T(s'),\ \forall\ s' \neq s$.

- a similar notation is used for other tables.

- $post(s_1, s_2, m)$ denotes $c = \langle location\_T, present\_T, mob\_T, ack\_T, k'\rangle$, with $k'(s_1, s_2) = k(s_1, s_2)\S\{m\}$, and $k'(s_i, s_j) = k(s_i, s_j)$, $\forall (s_i, s_j) \neq (s_1, s_2)$.

- $receive(s_1, s_2)$ denotes $c = \langle location\_T, present\_T, mob\_T, ack\_T, k'\rangle$, with $k(s_1, s_2) = \{m\}\S k'(s_1, s_2)$, and $k'(s_i, s_j) = k(s_i, s_j)$, $\forall (s_i, s_j) \neq (s_1, s_2)$.

Figure 8: Notation

A transition function maps a configuration c and a transition t to a new configuration c':

$$c \mapsto^t c',$$

Given a configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, five basic transitions are permitted:

```
migrate_agent$(s_1, s_2)$ :
  $s_1 \neq s_2 \;\wedge\; location\_T(s_1) = s_1 \;\wedge\; present\_T(s_1) \;\wedge\; ack\_T(s_1) = \mathsf{negative}$
  $\rightarrow \{$  $present\_T(s_1) := false$
          $post(s_1, s_2, \mathsf{agent}(mob\_T(s_1) + 1))$  $\}$

receive_agent$(s_1, s_2, l)$ :
  $first(k(s_1, s_2)) = \mathsf{agent}(l)$
  $\rightarrow \{$  $receive(s_1, s_2)$
          $location\_T(s_2) := s_2$
          $present\_T(s_2) := true$
          $mob\_T(s_2) := l$
          $ack\_T(s_2) := \mathsf{positive}(s_1, l)$  $\}$

ack_agent$(s_1, s_2, l)$ :
  $ack\_T(s_2) = \mathsf{positive}(s_1, l)$
  $\rightarrow \{$  $ack\_T(s_2) := \mathsf{negative}$
          $post(s_2, s_1, \mathsf{ack}(l))$  $\}$

receive_ack$(s_2, s_1, l)$ :
  $first(k(s_2, s_1)) = \mathsf{ack}(l) \;\wedge\; l > mob\_T(s_1)$
  $\rightarrow \{$  $receive(s_2, s_1)$
          $mob\_T(s_1) := l$
          $location\_T(s_1) := s_2$  $\}$

receive_ack2$(s_2, s_1, l)$ :
  $first(k(s_2, s_1)) = \mathsf{ack}(l) \;\wedge\; l \leq mob\_T(s_1)$
  $\rightarrow \{$  $receive(s_2, s_1)$  $\}$
```

Figure 9: Transitions for the Distributed Directory Service

Given a configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, three optimising transitions are permitted:

```
informt(s_1, s_2, s_3) :
  $location\_T(s_1) = s_2 \ \wedge\ s_1 \neq s_3 \ \wedge\ s_2 \neq s_3$
  $\rightarrow$ {  $post(s_1, s_3, \mathsf{inform}(mob\_T(s_1), s_2))$  }

receive_inform(s_2, s_1, s_3, l) :
  $first(k(s_2, s_1)) = \mathsf{inform}(l, s_3) \ \wedge\ l > mob\_T(s_1)$
  $\rightarrow$ {  $receive(s_2, s_1)$
        $mob\_T(s_1) := l$
        $location\_T(s_1) := s_3$  }

receive_inform2(s_2, s_1, s_3, l) :
  $first(k(s_2, s_1)) = \mathsf{inform}(l, s_3) \ \wedge\ l \leq mob\_T(s_1)$
  $\rightarrow$ {  $receive(s_2, s_1)$  }
```

Figure 10: Reducing Chains of Forwarding Pointers

where t is any of the eight allowed transitions. In a concise form, Figures 9 and 10 display the definitions of the transitions and the transition function.

In each rule of Figures 9 and 10, the conditions that appear to the left hand side of an arrow are guards that must be satisfied in order to perform the transition. The right-hand side denotes the configuration that is reached after transition. We assume that the guard evaluation and new configuration construction are performed atomically.

The first transition of Figure 9 is performed when an agent decides to migrate from s_1 to s_2. The present table on s_1 is set to false, and an agent message is posted between s_1 and s_2, with a mobility counter given as the successor of the mobility counter on s_1. Note that s_2, the destination of the agent, is only used to specify which communication channel the agent message must be enqueued into. The site s_1 does not need to be communicated this information, nor does it have to remember that site. In a real implementation, the agent message would also contain the complete agent state to be restarted by the receiver.

The second transition is concerned with s_2 handling an incoming agent(l) message from s_1. Tables are updated to reflect that s_2 is becoming the new agent's location, with l its current mobility counter. In addition, the table ack_T on s_2 is updated, since an acknowledgement has still to be sent back to s_1, with the current mobility counter l. At this point, a proper implementation would reinstate the agent state and resume its execution.

According to the third transition, if the content of an acknowledgement table is positive(s_1, l), an acknowledgement message ack(l) has to be sent from the current site to

s_1.

If a site s_1 receives an acknowledgement message with a mobility counter l, two cases are possible. If l is greater than the local mobility counter $mob\_T(s_1)$, then the location and mobility counter tables may be updated. Otherwise, the acknowledgement message is simply discarded, without updating any table.

According to the first rule of Figure 10, any site s_1 knowing that the agent is located on site s_2 may elect to communicate its knowledge to a third site s_3, passing along the value of the mobility counter the agent is known to have when it was on s_2. There is no need to inform oneself or the agent's location, hence the side-conditions $s_1 \neq s_3$ and $s_2 \neq s_3$. There is a similarity in the handling of `inform` and `ack` messages: tables are updated if the message timestamp is higher than the local mobility counter.

A guard deserves a further explanation: the acknowledgement table is required to be negative before the transition migrate_agent. Indeed, an agent is only allowed to leave a site after this site has emitted an acknowledgement message to the previous agent location (though the message does not have to be received before allowing migration). This constraint simplifies the algorithm (and its proof) because a site has to remember at most one site to acknowledge (as opposed to an unbounded set of sites). We do not see this condition as significantly changing the performance of this algorithm.

4 Message Routing

In order to define the store-and-forward routing algorithm previously presented, we need to extend the abstract machine. A new message constructor user creates messages to be routed to an agent (Figure 11). We use a bag in order to accumulate messages that must be delivered to an agent in transit; it is sufficient to use a bag, as opposed to a queue, as the algorithm does not preserve message order.

$$\begin{array}{rcll} \mathcal{M} & = & \mathsf{user} : \mathit{Content} \to \mathcal{M} \ \mid \ \ldots & \text{(Messages)} \\ \mathcal{P} & = & \mathcal{S} \to \mathit{BagOf}(\mathcal{M}) & \text{(Pool of Messages)} \\ & & & \\ \mathcal{C} & = & \mathcal{LT} \times \mathcal{PT} \times \mathcal{MT} \times \mathcal{AT} \times \mathcal{P} \times \mathcal{K} & \text{(Configurations)} \end{array}$$

Notation: $pool\_T \in \mathcal{P}$.

Figure 11: Extended State Space

Transitions for the message routing algorithm appear in Figure 12. These transitions use the information provided by the distributed directory service in order to route messages to the mobile agent. The only information that needs to be made available is the contents of the location and present tables on each site. The message router does not modify these tables, but simply reads their contents.

The first rule deals with the sending of a message to the mobile agent from a site s. (In practice, such an action would be initiated by another agent, which requests s to send the message on its behalf.) The message is simply added to the bag of user messages to be processed on that site.

```
send_user_msg$(s, content)$ :
  $\rightarrow$ {  $pool\_T(s) := pool\_T(s) \cup \{\mathsf{user}(content)\}$  }

deliver_user_msg$(s, content)$ :
  $first(pool\_T(s)) = \mathsf{user}(content) \wedge location\_T(s) = s \wedge present\_T(s)$
  $\rightarrow$ {  $receive(pool\_T(s))$
        $deliver(content)$  }

forward_user_msg$(s_1, s_2, content)$ :
  $first(pool\_T(s_1)) = \mathsf{user}(content) \wedge location\_T(s_1) = s_2 \wedge s_1 \neq s_2$
  $\rightarrow$ {  $receive(pool\_T(s_1))$
        $post(s_1, s_2, \mathsf{user}(content))$  }

receive_user_msg$(s_1, s_2, content)$ :
  $first(k(s_1, s_2)) = \mathsf{user}(content)$
  $\rightarrow$ {  $receive(k(s_1, s_2))$
        $pool\_T(s_1) := pool\_T(s_1) \cup \{\mathsf{user}(content)\}$  }
```

Figure 12: Transitions for the Message Router

In the second rule, the directory service indicates that the agent is present on s. Any message sitting in the pool of messages may be delivered directly to the agent.

If the agent is not present on a site s_1, any user message waiting to be processed on s_1 may be forwarded to the agent's location s_2, as contained in the location table of s_1.

Finally, the fourth rule of Figure 12 simply adds incoming user messages to the pool of messages, which remain there until they become processed.

The initial configuration is defined as follows. We assume that the agent is known to be at a given site *origin* with a mobility counter set to 0. Present tables are false except for the origin site. Acknowledgement tables are all negative. Communication queues and pools of messages are all initially empty. Formally, the initial configuration c_i is defined by the tuple $\langle location\_T_i, present\_T_i, mob\_T_i, ack\_T_i, pool\_T_i, \mathcal{K}_i\rangle$, where:

$$\begin{array}{rcl}
location\_T_i & = & \lambda s.origin \\
present\_T_i & = & \lambda s.s = origin \\
mob\_T_i & = & \lambda s.0 \\
ack\_T_i & = & \lambda s.\mathsf{negative} \\
pool\_T_i & = & \lambda s.\emptyset \\
\mathcal{K}_i & = & \lambda s_1 \lambda s_2.\emptyset.
\end{array}$$

A configuration c is said to be *legal* if there is a sequence of transitions $t_1, t_2, \ldots, t_n$ such that c is reachable from the initial configuration:

$$c_i \mapsto^{t_1} c_1 \mapsto^{t_2} c_2 \ldots \mapsto^{t_n} c.$$

5 Algorithm Correctness

The purpose of this section is to establish the correctness of both algorithms. We expect the distributed directory service to tell us where the mobile agent is currently located, which is a form of *safety property*; we also expect this information to eventually be updated as the agent migrates, which is a *liveness property*.

Similar properties are anticipated from the message router. We would expect messages to be routed in the agent's direction and to be delivered eventually. The formalisation will have to refine this statement, as we cannot guarantee that messages can be delivered to a runaway mobile agent; however, the property holds once the agent stops migrating.

Section 5.1 investigates the correctness of the directory service, whereas Section 5.2 deals with the message router. The essence of the proof may be summarised as follows. Forwarding pointers form a graph of sites. We establish that this graph has no cycle, and is in fact a tree with a unique root: the site where the agent is. Therefore, the routing information provided by forwarding pointers necessarily leads to the agent.

5.1 Directory Service

The proof of the directory service has been developed using the proof assistant Coq; it required approximately 11000 lines of tactic invocation for the formalisation of the algorithm and the derivation of its correctness. We use a library initially developed for proving the correctness of a distributed reference counting algorithm [23]. In this section, we present the key properties we established, and the proof details may be found in [22].

First, we define some concepts, which we use in the proof. When a mobile agent is not in transit, we refer to the site currently hosting it as its *host*.

Definition 1 (Agent host)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k \rangle$, *and for any site* s, $agent\_host(c, s)$ *holds if* $location\_t(s) = s$ *and* $present\_T(s)$. □

We will establish that a given agent, at any time, may be in transit at most once. We define $agent\_count(l, k)$ as the number of messages $\mathsf{agent}(l)$ in transit. We use the symbol # to denote the cardinality of a set.

Definition 2 (Agent Count)

$$agent\_count(l, k) = \#\{\mathsf{agent}(l) \in k(s_1, s_2),\ \textit{for any}\ s_1, s_2 \in \mathcal{S}\}.$$

□

A vital invariant has to be established at an early stage. It states that an agent can be either situated at a site or in transit. Furthermore, the mobility counter associated with the agent is greater than any other instance of the counter in the system. Such a property is formally established in Lemma 3, where a disjunction describes the two possible cases. For instance, the left-hand disjunct considers the existence of a site s being the agent host. This agent host is unique, and there is no instance of the agent in transit. In addition, any instance of a mobility counter l_1, such as in $\mathsf{ack}(l_1)$, is smaller than the mobility counter $mob\_T(s)$ on s.

Lemma 3 (Situated or Migrating)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, such that $c_i \mapsto^* c$:

$$
\begin{array}{l}
\{\ \exists s \in \mathcal{S}\ |\\
\quad (\ agent\_host(c, s)\\
\quad\quad \wedge\ \forall s_i \in \mathcal{S},\ \ agent\_host(c, s_i) \rightarrow s = s_i\\
\quad\quad \wedge\ \forall s_1, s_2 \in \mathcal{S}, \forall l \in Z,\ \ \mathsf{agent}(l) \notin\ \ k(s_1, s_2)\\
\quad\quad \wedge\ \forall s_1, s_2 \in \mathcal{S}, \forall l_1 \in Z,\ \ ack\_T(c, s_1) = \mathsf{positive}(s_2, l_1) \rightarrow s_1 = s\\
\quad\quad \wedge\ \forall s_1, s_2 \in \mathcal{S}, \forall l_1 \in Z,\ \ ack\_T(c, s_1) = \mathsf{positive}(s_2, l_1), \rightarrow l_1 = mob\_T(c, s)\\
\quad\quad \wedge\ \forall s_1, s_2 \in \mathcal{S}, \forall l_1 \in Z,\ \ \mathsf{ack}(l_1) \in k(s_1, s_2) \rightarrow l_1 \leq mob\_T(c, s)\\
\quad\quad \wedge\ \forall s_i \in \mathcal{S},\ \ mob\_T(c, s_i) \leq mob\_T(c, s)\\
\quad\quad \wedge\ \forall s_1, s_2, s_3 \in \mathcal{S}, \forall l_1 \in Z,\ \ \mathsf{inform}(l_1, s_3) \in k(s_1, s_2)\ \ \rightarrow\ \ l_1 \leq mob\_T(c, s)\quad)\ \}\\
\vee\\
\{\ \exists (l, s_a, s_b) \in (Z \times \mathcal{S} \times \mathcal{S})\ |\\
\quad\quad (\mathsf{agent}(l) \in k(s_a, s_b)\\
\quad\quad\quad \wedge\ agent\_count(l, k) = 1\\
\quad\quad\quad \wedge\ \forall li \in Z,\ \ l_i \neq l \rightarrow agent\_count(l_i, k) = 0\\
\quad\quad\quad \wedge\ \forall s_2 \in \mathcal{S},\ \ \neg agent\_host(c, s_2)\\
\quad\quad\quad \wedge\ \forall s_1 \in \mathcal{S},\ \ ack\_T(c, s_1) = \mathsf{negative}\\
\quad\quad\quad \wedge\ \forall s_i \in \mathcal{S},\ \ mob\_T(c, s_i) < l\\
\quad\quad\quad \wedge\ \forall s_1, s_2 \in \mathcal{S}, \forall l_1 \in Z,\ \ \mathsf{ack}(l_1) \in k(s_1, s_2) \rightarrow\ \ l_1 < l\\
\quad\quad\quad \wedge\ \forall s_1, s_2, s_3 \in \mathcal{S}, \forall l_1 \in Z,\ \ \mathsf{inform}(l_1, s_3) \in k(s_1, s_2) \rightarrow l_1 < l\quad)\ \}.
\end{array}
$$

□

Proof of Lemma 3
The proof appears in file `invariant0.v` [22] and proceeds by induction on the legal transitions, and by a case analysis of the different transitions. □

An essential aspect of the algorithm is to ensure that tables are not updated with older information, in order to prevent the formation of cyclic routes, as illustrated in Section 2. To this end, we designed rules receive_ack and receive_inform with a guard $l > mob\_T(s_1)$. As a result, we can establish that a site mobility counter is never decreasing.

Lemma 4 (Non Decreasing Counters)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, and for any configuration $c' = \langle location\_T', present\_T', mob\_T', ack\_T', k'\rangle$, such that $c_i \mapsto^* c \mapsto^t c'$, for any site s:

$$mob\_T(s) \leq mob\_T'(s).$$

□

Proof of Lemma 4
The proof appears in file `invariant2.v` and proceeds by induction on the legal transitions, and by a case analysis of the different transitions. □

We can regard the contents of the location table as a trail of forwarding pointers that lead to the agent's location. We can formally define a relation between sites that capture this notion of forwarding pointer. We say that s_2 is the *parent* of s_1, if $location\_T(s_1) = s_2$. However, agent migration is not atomic: the location table of the agent's previous location is only updated after transitions $\mathsf{ack\_agent}$ and $\mathsf{receive\_ack}$ are performed. Therefore, we define a *parent* relation, which is stable when these transitions are executed.

Definition 5 (Parent)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, *for any sites* s_1, s_2, $parent(c, s_1, s_2)$ *holds if:*

$$\begin{array}{lll}
 & location\_T(s_1) = s_2 \;\wedge\; s_1 \neq s_2 & (1) \\
or & \exists l \in Z,\ location\_T(s_1) = s_1 & (2) \\
 & \quad \wedge\ \neg present\_T(s_1) & \\
 & \quad \wedge\ \mathsf{ack}(l) \in k(s_1, s_2) & \\
 & \quad \wedge\ mob\_T(s_1) = l - 1 & \\
or & \exists l \in Z,\ location\_T(s_1) = s_1 & (3) \\
 & \quad \wedge\ \neg present\_T(s_1) & \\
 & \quad \wedge\ ack\_T(s_2) = \mathsf{positive}(s_1, l) & \\
 & \quad \wedge\ mob\_T(s_1) = l - 1. &
\end{array}$$

□

The second case of the definition considers an ack message in transit, while the third case is about acknowledgement messages remaining to be sent.

As we follow forwarding pointers, we expect to get closer to the mobile agent's position. This intuition is partially captured by the following lemma, which states that mobility counters are increasing along edges of the parent relationship.

Lemma 6 (Increasing Parent)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$ *such that* $c_i \mapsto^* c$, *for any sites* s_1, s_2, *if* $parent(c, s_1, s_2)$ *holds, then:*

$$mob\_T(s_1) \leq mob\_T(s_2).$$

□

Proof of Lemma 6
The proof appears in file `invariant3.v` and proceeds by induction on the legal transitions, and by a case analysis of the different transitions. □

Knowing that the agent owns the highest mobility counter, we would like to prove that there is a finite number of hops between any host and the agent. However, Lemma 6 is not as strong as we might have wished, because inequality is not strict. Consequently, we still have to prove that the relationship *parent* defines a tree and not a graph.

Two steps are required to establish such a result. First, we can prove that a site has at most one parent.

Lemma 7 (Unique Parent)
For any configuration c accessible from the inital configuration $c_i \mapsto^* c$, for any sites $s_1, s_2, s_3 \in \mathcal{S}$, if $parent(c, s_1, s_2)$ and $parent(c, s_1, s_3)$ hold, then $s_2 = s_3$. □

Proof of Lemma 7
The proof appears in file `invariant4.v` and proceeds by induction on the legal transitions, and by a case analysis of the different transitions. □

Second, we can show that two sites, connected by an edge of the parent relationship, and having the same mobility counter, have location tables with a very specific contents.

Lemma 8
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, for any site s_1, s_2, such that $parent(c, s_1, s_2)$ and $mob\_T(s_1) = mob\_T(s_2)$, then the following equalities hold:

$$location\_T(s_1) = s_2 \text{ and } location\_T(s_2) = s_2.$$

□

Proof of Lemma 8
In order to establish this result, we proceed by induction on the legal transitions, and by a case analysis of the different transitions. Details appear in file `invariant4.v`. □

Intuitively, Lemma 8 means that if two sites s_1, s_2 connected by the parent relation have the same mobility counter, then the agent is still located at s_2, or if it is not located at s_2, no acknowledgement message has been received by s_2 yet. Once the acknowledgement message is received by s_2, its mobility counter will increase. We now have established all the properties required to prove that the parent relationship does not create cycles. Let us define *ancestor* as the transitive closure of parent. The absence of cycles is stated as follows.

Lemma 9 (Absence of Cycle) *For any configuration c accessible from the inital configuration $c_i \mapsto^* c$, for any sites $s_1, s_2 \in \mathcal{S}$, if $ancestor(c, s_1, s_2)$ holds, then $parent(c, s_2, s_1)$ does not hold.* □

Proof of Lemma 9
Should $parent(c, s_2, s_1)$ hold, we would be able to derive the following inequalities

$$mob\_T(s_1) \leq mob\_T(s_2) \leq mob\_T(s_1),$$

by repeatedly using Lemma 6. This implies the following equalities:

$$mob\_T(s_1) = mob\_T(s_2) = mob\_T(s_1),$$

from which we can conclude that $s_1 = s_2$ by Lemma 8; we reach a contradiction because we can prove that the parent relation is not reflexive. Details can be found in file `invariant5.v`. □

The absence of cycles is not sufficient to guarantee the algorithm safety. Indeed, we want to make sure that forwarding pointers do lead to the agent's location. We formally define the *terminal site* as the agent host if the agent is not in transit, or its previous site if the agent is in transit.

Definition 10 (Terminal Site)
For any configuration c, the terminal site*, written $terminal\_site(c)$, is a site s such that $agent\_host(c, s)$ holds if the agent is not in transit, or such that there exists a message* agent(l) *in a message queue from s.* □

We can establish that the *parent* relationship is well-founded. Therefore, it is possible to construct by a fixed point, a function that associates any site with a site without parent, which can be proved to be the terminal site. As a result, this guarantees that, from any site, forwarding pointers lead to the unique terminal site.

In order to show that *parent* is a well-founded relation, we need to prove that there is a measure that *strictly* decreases along edges of the relation. Intuitively, for a site s, this measure can be defined as the difference between the mobility counter of the terminal site and the mobility counter of s. Lemma 6 has however established that mobility counters are not strictly increasing; therefore, we have to consider the existence of acknowledgement messages in transit. Such a measure appears in Definition 11.

Definition 11 (Hops)
For any configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$, for any site s, the measure $Hops(c, s)$ is defined as follows:

- *If there exists $s_2 \in \mathcal{S}$, such that $parent(c, s, s_2)$ holds, then:*
  - *if $location\_T(s) = s_2$, $Hops(c, s) = mob\_T(terminal\_site(c)) - mob\_T(s) + 1$.*
  - *if $location\_T(s) \neq s_2$, $Hops(c, s) = mob\_T(terminal\_site(c)) - mob\_T(s)$.*
- *Otherwise: $Hops(c, s) = 0$.*

□

If the mobile agent was never coming back to a previous location, and if inform messages were never sent, the measure *Hops* would exactly be the number of hops required to reach a mobile agent. In reality, this measure is a maximal bound on the number of hops. We can prove that this measure is strictly decreasing:

Lemma 12 (Strictly Decreasing)
For any configuration c, for any sites $s_1, s_2 \in \mathcal{S}$, if $parent(c, s_1, s_2)$ holds then:

$$Hops(c, s_2) < Hops(c, s_1).$$

□

Proof of Lemma 12
We proceed by a case analysis of the hypothesis $parent(c, s_1, s_2)$. Details of the proof can be found in file `invariant6.v`. □

Since we can decide whether a site s has a parent or not, and since we have defined a measure that is strictly decreasing (Lemma 12), we can prove that the relation *parent* is well-founded, and that we can construct a function that associates any site with a site without any parent.

Lemma 13
There exists a function root that associates any site with a site without parent. Formally, there exists a function root : $\mathcal{S} \to \mathcal{S}$, *such that for any* $s \in \mathcal{S}$, *there is no* $s_i \in \mathcal{S}$ *such that* $parent(c, root(s), s_i)$. □

Proof of Lemma 13
The proof, which can be found in file `invariant6.v`, relies on Coq predefined notion of well-founded relation, and the ability to define a function over a well-founded relation by a fixed point. □

Now, it remains to prove that the site without successor is unique, and is actually equal to the terminal site of a configuration. First, uniqueness is derived by the following lemma, which states that there is only one site without parent.

Lemma 14 (Unique Orphan)
For any legal configuration c, $\forall s_1, s_3 \in \mathcal{S}$, *if there is no* $s_2 \in \mathcal{S}, parent(c, s_1, s_2)$, *and if there is no* $s_4 \in \mathcal{S}, parent(c, s_3, s_4)$, *then* $s_1 = s_3$. □

Proof of Lemma 14
The proof appears in file `invariant6.v` and proceeds by induction on the legal transitions, and by a case analysis of the different transitions. □

We can therefore derive the safety property:

Theorem 15 (Safety)
For any configuration c, from any site, the parent relationship leads to the terminal site of c. □

Proof of Theorem 15
We can prove that the terminal site has no parent, and using Lemma 14, we conclude that it is equal to the value of the *root* function for any site s (Lemma 13). Details of the proof appear in file `invariant6.v`. □

The *parent* relationship can only be decided by examining the whole distributed system; it is not convenient to implement, because it requires us to know if there are ack messages in transit, or if some acknowledgement tables are not negative. However, once acknowledgement messages have been processed, the parent relationship is given by the contents of the location table. Therefore, the algorithm will be implementable, only if it has the *liveness property*, which ensures that location tables get updated to reflect the parent relationship.

First, we establish that a finite amount of transitions can be performed from any legal configuration, if we prevent the agent from migrating and new `inform` messages from being sent. For this purpose, we can define a measure that is a function of the number of messages in transit and the contents of acknowledgement tables.

Definition 16 (Configuration Measure)
Let us consider a configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k\rangle$. *The measure of configuration c, written as* $|c|$, *is defined as follows.*

$$|c| \;=\; \#(\mathsf{ack}(l)\; \in k(s_i, s_j),\; \forall s_i, s_j)$$

$$\begin{array}{ll} + & \#(\mathsf{inform}(l,s) \ \in k(s_i,s_j),\ \forall s_i,s_j) \\ + & 2\ \#(s \mid ack\_T(s) \neq \mathsf{negative}) \\ + & 3\ \#(\mathsf{agent}(l) \ \in k(s_i,s_j),\ \forall s_i,s_j) \end{array}$$

□

Lemma 17 (Decreasing Measure)
For any configurations c, c', for any transition t different from informt *and* migrate_agent*, such that $c \mapsto^t c'$,*

$$|c| > |c'|.$$

□

Proof of Lemma 17
The proof is by case analysis on the possible transitions. The only transition to create a new message is ack_agent, which still decreases the measure as it diminishes the number of non negative acknowledgement tables. Details are found in file `invariant7.v`. □

Consequently, we can derive the termination of the algorithm.

Theorem 18 (Termination) *For any legal configuration, all transition paths that do not involve* informt *and* migrate_agent *transitions terminate.* □

Proof of Theorem 18
We can again define a well-founded relation based on the transition relation. On that domain, we can construct by fixed point a function that associates any configuration to a configuration without successor (cf. file `invariant7.v`). □

We can also prove that, if there is a message which is the first of a queue of messages, there exists a transition of the abstract machine that consumes that message. Consequently, if we assume that message delivery, machine transitions, and sending of `inform` messages are fair, then location tables will eventually be updated, which proves the liveness of the algorithm.

We have modelled communication channels by queues of messages. We can show that message order is not a requirement of the algorithm, by introducing an extra rule that rearranges message order in a queue; we have established that such a rule preserves the safety property (proof details may be found at [22]).

$$\begin{array}{l} \mathsf{out\_of\_order}(s_1,s_2,m,q_1,q_2,q_3,q_4): \\ \quad k(s_1,s_2) = q_1 \ \S\ \{m\} \ \S\ q_2 \quad \wedge \quad q_1 \S q_2 = q_3 \S q_4 \\ \quad \rightarrow \ \{ \quad k(s_1,s_2) := q_3 \ \S\ \{m\} \ \S\ q_4 \ \} \end{array}$$

5.2 Message Router

The safety of the message router is an immediate consequence of the safety of the directory service: indeed, according to rule deliver_user_msg, user messages are only delivered to an agent on the agent host, known to be the only location where the agent is, when it is not in transit.

It remains to establish the liveness property of the message router, according to which any user message is eventually delivered. As previously indicated, this property is too strong, because messages cannot be delivered to a runaway agent, i.e. an agent that migrates as quickly as messages are forwarded. Therefore, as in Section 5.1, we need to consider configurations where agent migration is not allowed. We can establish that, once the agent has become stationary, if no new user or inform messages are sent, it takes a finite number of transitions to deliver all user messages.

We update Definition 16 in order to take user messages into account.

Definition 19 (Configuration Measure 2)
Let us consider a configuration $c = \langle location\_T, present\_T, mob\_T, ack\_T, k, pool\_T\rangle$. The measure of configuration c, written as $|c|_u$, is defined as follows.

$$\begin{array}{rcl} |c|_u & = & |c| \\ & + & \sum_{s_i,s_j}(2Hops(c,s_j)+1, \quad \textit{if}\ \mathsf{user}(content)\ \in k(s_i,s_j)) \\ & + & \sum_{s_i}(2Hops(c,s_i), \quad \textit{if}\ \mathsf{user}(content)\ \in pool\_T(s_i)) \end{array}$$

□

In Definition 19, the weight of a message in transit to a site s_j is one plus the weight it would have if it was in the pool of messages of s_j. The weight of a message in a pool is twice the value of the function *Hops* for the site. The measure is such that any transition related to a user message decreases the overall measure strictly.

Lemma 20 (Decreasing Measure2)
For any configurations c, c', for any transition t different from informt, migrate_agent, send_user_msg, *such that $c \mapsto^t c'$,*

$$|c|_u > |c'|_u.$$

□

Proof of Lemma 20
Lemma 17 deals with $|c|$, whereas we have to proceed by a case analysis on the possible transitions of Figure 12. □

Following the same argument as in the previous section, we can derive the termination of the algorithm.

Theorem 21 (Termination 2) *For any legal configuration, all transition paths that do not involve* informt, migrate_agent, *and* send_user_msg *transitions terminate.* □

Proof of Theorem 21
The proof technique is exactly the one used for Theorem 18. □

It is also true that if there is a user message that is the first message in a queue, there exists a transition, namely receive_user_msg, which consumes that message. Since the directory service has the liveness property, a site will eventually become the agent's host or a forwarder; therefore, any user message in a *pool_T*, will be either delivered to a local agent or forwarded to another location. Consequently, if we assume that message delivery, transitions, sending of `inform` messages, and the directory service are fair, then all user messages will eventually be processed, which gives us the liveness property.

In this section, we have established the correctness of the routing algorithm. Let us remember that correctness does not involve in-order message delivery, as the potentially different routes taken by messages do not guarantee any arrival order. The current liveness property requires us to consider a mobile agent that has stopped migrating. If we introduce a notion of time, we could define the speed of agent migration and the speed of message forwarding; a liveness property in this context would establish the maximum agent speed in terms of message forwarding speed, in order to guarantee delivery.

6 Discussion and Related Work

The initial configuration of the algorithm requires each site to know where a mobile agent was created. While such a condition is acceptable for a theoretical algorithm, some ingenuity is required to make it practical. A centralised solution would involve a yellow page service that would remember where agents are created. A distributed version could embed the creation site of an agent in its name: the agent's name would automatically provide a default routing information. In addition, sites do not need to know about all agents by default; routing tables may be built incrementally when references to agents are exchanged or when agents migrate.

If our algorithm was used on an Internet scale, we would have to provide a mechanism for clearing routing tables in order to avoid their overflow. Two mechanisms are worth investigating: time-outs and reference counting. *(i)* After a specified timeout, a site could clear its unused entries. Unfortunately, the site would be unable to route messages that are still in transit. In fact, we would be in a situation that is similar to a site failure, which we discuss below. *(ii)* Another solution is based on the observation that clearing routing tables is equivalent to the distributed termination problem [31]. A site is allowed to clear its entries if it can prove that no other site will ever again forward information to it. This may be implemented using a distributed reference counting algorithm [21, 23, 30, 31]: once the distributed reference counter becomes null, the entry may be removed.

The interaction between distributed reference counting and message routing still needs to be investigated. In [21, 23], we describe a distributed reference counting algorithm that has the ability to reorganise the diffusion of pointer, thereby reducing the distributed state and possible dependencies between sites. A useful property that would have to be established is the "table liveness", by which routing information is cleared once it is no longer used.

Shapiro *et al.* [30] presented an algorithm for garbage collecting mobile objects. We have identified three different tasks required for that activity: distributed reference counting, distributed directory service, and message routing. Like Shapiro, we have designed our algorithms to reduce chains of dependencies: we want to investigate their interaction further, and in particular we wish to study how reference counters can be made mobile. Shapiro's technique based on the notion of stub/scion pair was further extended to support communication with a current support station in wireless communications [3].

We began investigating how a distributed state, such as the one resulting from a distributed garbage collector, could be made resilient to hosts disconnections. At first, we consider graceful disconnections and not failures [29]; sites that wish to disconnect have to cooperate with other sites, so as to migrate objects that are located on them, using a mobility protocol as the one described in this paper. Then, the distributed state has to be modified, so that no pointer ever refers to the disconnecting sites.

This naturally brings us to the topic of tolerance to failure. Different failures may occur in the context of mobile agents. Communication failure may result in the loss of messages or in the loss of an agent in transit. Failure of a site may account for the loss of the agents it is hosting, or the loss of routing information. Transport protocols may be used to ensure reliable message communications. Mishra, Jiang and Yang [20] describe a watchdog mechanism responsible for ensuring the successful migration of an agent to a new site, and for recovering an agent in case of a site failure. Therefore, it remains to address the failure of an intermediate site: a solution to this problem has to offer alternate routes to messages that would have normally been routed to the site that failed.

We are currently investigating the use of inform messages for duplicating routing information. The idea of this approach is to propagate the new location of an agent to the n previous different sites it visited. This would ensure that, at any time, several routes to an agent would be known, which would tolerate a maximum number of intermediate sites failures. We are studying the correctness of this algorithm and its use to route messages.

Inform messages are critical in the algorithm to reduce chains of forwarding pointers: they may reduce communications cost and inter-sites dependencies. The ideal strategy to send these messages is dependent on the type of distributed system and the application using mobile agents. A range of solutions is possible and two extremes of the spectrum are easily identifiable. In a lazy strategy, the recipient of a message informs the emitter, when the recipient observes that that the emitter has out-of-date routing information. In such a strategy, tables are only updated when user messages are sent. In an eager strategy, every time a mobile agent migrates, the new destination broadcasts the new agent position to all others sites; such a solution is clearly not acceptable for the Internet. Other solutions within that spectrum are possible, such as back propagating inform messages along the opposite route taken by a user message.

An important motivation for mobility [7] is the presence of firewalls that prevent direct communications to resources from outside the domain they protect. In order to circumvent that problem, mobile agents typically have to migrate to a specific host, a "domain entry point", where security checks [26] may be run; if checks are successful, the mobile agent is granted the right to migrate inside the domain protected by the firewall. Our algorithm may still be used in the presence of firewalls: it is able to route messages through a firewall, if accepted by its security policy. Indeed, the "domain entry point" can also maintain routing tables and implement our algorithm. It will act as a message router from one side of the firewall to the other. Inform messages can still be propagated, but not across the firewall; their effect will be to reduce chains of forwarding pointers on either side of the "domain entry point". A similar mechanism, called gateway, is used by Baggio and Piumarta [3] to prevent short cutting of SSP chains in wireless communications.

If this algorithm was adapted for a lower-level network layer, a bound on the number of message to be kept would have to be imposed. This would require, either *(i)* discarding messages, which would make the protocol unreliable, and would require an end-to-end reliable transport protocol, *(ii)* back-propagating flow control to the emitter, to prevent it from emitting further messages.

The Internet Engineering Task Force has been investigating a mobility layer for the version 6 of the IP protocol (IPv6) [5, 15, 28, 9]. An essential motivation in the design of IPv6 is preserving compatibility with the current Internet Protocol, where hosts are unaware of mobile hosts. Therefore, for each mobile host, there exists a home agent acting as a proxy to tunnel messages to the mobile host. Research about IPv6 consists of optimising communication within an infrastructure that remains compatible with the current IP organisation. There are two key differences between IPv6 and our algorithm. First, we do not rely on a fixed home agent to forward messages to a mobile agent. Second, our algorithm uses names as a location independent identifier for a mobile agent; on the contrary, an IP address serves dual purposes: "from the transport and application layer perspective, it serves as an end-point identifier, and at the network layer, the IP address is used as a routing directive" [5]. It would be interesting to investigate if our algorithm could be used in the context of IPv6; this requires further research beyond the scope of this paper.

There exist several mobile agent systems such as Agent TCL [12], Ajanta [32], APRIL [18], Ara [27], Tacoma [14]. To the best of our knowledge, APRIL is the only one to provide a uniform mechanism to communicate with mobile agents. Indeed, APRIL and the derived library ICM[5], provide a store-and-forward communication model for fixed and mobile agents. Agents are identified by globally unique names; each name contains the location where an agent is created, and is used as routing information. For a mobile agent, the name contains further routing information, such as the latest visited site and home agent. By default, communication is attempted with the latest known agent's location; if it fails, the communication is established with the home base, which forwards messages to the agent. This communication model is lower-level than our proposed algorithm because it is the programmer's responsibility to maintain the latest known position of an agent, and suitable routing information in their name. In addition, the ICM has a mechanism by which a mobile agent may leave a forwarding pointer, which will be used to forward messages directly, without involving the home agent. This technique is similar to the one described here, but no mention is made of a mechanism to avoid cyclic routing.

7 Conclusion

In this paper, we have presented two algorithms that can be used in the infrastructure necessary to support mobile agents: a distributed directory service tracks where a mobile agent is, and a message router transparently routes messages to a mobile agent. A

[5]Many of these ideas were incorporated in the FIPA definition of mobile agents [10].

key aspect in the design of both algorithms is the absence of any fixed resource, which makes the algorithm suitable for massively distributed systems, such as the amorphous computer or the ubiquitous/pervasive computing environment. The safety and liveness of both algorithms were established, and encoded in the proof assistant Coq.

Several research topics derived from this paper are being investigated. These algorithms are currently being implemented as part of a distributed agent infrastructure [24]; we intend to integrate them with our model of distributed resources [25]. We also wish to evaluate the performance of the algorithms using a simulator for the amorphous computer [2]. Finally, we plan to investigate how inform messages could systematically be used to duplicate routing information, so as to provide fault-tolerant versions of these algorithms; further work is also required to guarantee bounded resources at routing nodes, which would make this algorithm suitable for network routing.

8 Acknowledgement

Part of this research was undertaken during the author's stay at Ecole Normale Supérieure, Lyon. Luc Moreau would like to thank Luc Bougé for very fruitful discussions and for making this visit possible, as well as Ecole Normale Supérieure, Lyon and INRIA Rhones-Alpes for their support. The Coq proof of the correctness of the algorithm uses a library of theorems proved in collaboration with Jean Duprat. Thanks to David DeRoure, Christian Queinnec, Tim Chown, Danius Michaelides, and Mark Thompson for useful comments and discussions on the subject of mobility.

References

[1] Matin Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: the Spi Calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zürich, April 1997*, 1997.

[2] S. Adams and David C. DeRoure. A Simulator for an Amorphous Computer. In *Proceedings of the 12th European Simulation Multiconference (ESM'98)*, Manchester, UK, June 1998.

[3] Aline Baggio and Ian Piumarta. Mobile Host Tracking and Resource Discovery. In *7th ACM SIGOPS European Workshop*, Connemara, Ireland, September 1996. http://www-sor.inria.fr/publi/MHTRD_sigops96.html.

[4] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Giménez, H. Herbelin, G. Huet, C. Mu noz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.

[5] Pravin Bhagwat, Charles Perkins, and Satish Tripathi. Network Layer Mobility: an Architecture and Survey. *IEEE Personal Communication*, 3(3), June 1996.

[6] Krishna Bharat and Luca Cardelli. Migratory Applications. In *Mobile Object Systems: Towards the Programmable Internet*, pages 131–149. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.

[7] Luca Cardelli. Abstractions for Mobile Computation. In Jan Vitek and Christian Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, volume 1603 of *Lecture Notes in Computer Science*, 1999.

[8] Luca Cardelli and Andrew Gordon. Mobile Ambients. In *Foundations of Software Science and Computational Structures (ETAPS'98)*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155, May 1998.

[9] Stuart Cheshire and Mary Baker. Internet Mobility 4x4. In *Proceedings of ACM SIGCOMM'96*, Stanford, California, August 1996.

[10] Jonathan Dale and Francis G. McCabe. Agent Management Support for Mobility. Fipa'98 draft specification, Fujitu Laboratories of America, 1998.

[11] Cédric Fournet and Georges Gonthier. The Reflexive Chemical Abstract Machine and the Joinf-Calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 21-24 1996. ACM.

[12] Robert S. Gray. Agent Tcl: A Flexible and Secure Mobile-Agent System. In Mark Diekhans and Mark Roseman, editors, *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, Monterey, California, July 1996. http://www.cs.dartmouth.edu/ ∼agent/papers/index.html.

[13] Michael Hicks, Pankay Kakkar, Jonathan T. Moore, Car A. Gunter, and Scott Nettles. Network Programming Using PLAN. In *Worshop on Internet Programming Languages*, Loyola University, Chicago, May 1998.

[14] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating System Support for Mobile Agents. In *5th IEEE Workshop on Hot Topics in Operating Systems*, Orcas Island, Wa, USA, 1995. Also available as Technical Report TR94-1468, Department of Computer Science, Cornell University. http://www.cs.uit.no/DOS/Tacoma/Publications.html.

[15] David B. Johnson and Charles Perkins. Mobility Support in IPv6. Internet draft, IETF Mobile IP Working Group, 1999. draft-ietf-mobileip-ipv6-09.txt.

[16] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, December 1995.

[17] General Magic. Telescript Technology: Mobile Agents. http://www.genmagic.com/ Telescript/Whitepapers/ wp4/whitepaper-4.html, 1996.

[18] F. G. McCabe and K. L. Clark. APRIL - Agent Process Interaction Language. In *Proc. of ECAI'94 Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1995.

[19] F. H. McCabe. InterAgent Communications Reference manual. Technical report, Fujitu Laboratories of America, 1999.

[20] S. Mishra, X. Jiang, and B. Yang. Providing Fault Tolerance to Mobile Intelligent Agents. In *Proceedings of the ISCA 8th International Conference on Intelligent Systems*, Denver, June 1999.

[21] Luc Moreau. A Distributed Garbage Collector with Diffusion Tree Reorganisation and Object Mobility. In *Proceedings of the Third International Conference of Functional Programming (ICFP'98)*, pages 204–215, September 1998. Also in *ACM SIGPLAN Notices*, 34(1):204-215, January 1999.

[22] Luc Moreau. Distributed Directory Service and Message Router for Mobile Agents: the Constructive Proof in COQ. Available from `http://www.staff.ecs.soton.ac.uk/~lavm/coq/mobility/`, October 1999.

[23] Luc Moreau and Jean Duprat. A Construction of Distributed Reference Counting. Technical Report RR1999-18, Ecole Normale Supérieure, Lyon, March 1999.

[24] Luc Moreau, Nick Gibbins, David DeRoure, Samhaa El-Beltagy, Wendy Hall, Gareth Hughes, Dan Joyce, Sanghee Kim, Danius Michaelides, Dave Millard, Sigi Reich, Robert Tansley, and Mark Weal. An Agent Framework for Distributed Information Management. Technical Report ECSTR M99/4, Multimedia Research Group, University of Southampton, 1999.

[25] Luc Moreau and Christian Queinnec. Design and Semantics of Quantum: a Language to Control Resource Consumption in Distributed Computing. In *Usenix Conference on Domain-Specific Languages (DSL'97)*, pages 183–197, Santa-Barbara, California, October 1997.

[26] George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, Paris, France, January 1997.

[27] H. Peine and T. Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Proc. of the First International Workshop on Mobile Agents MA'97*, number 1219 in Lecture Notes in Computer Science, Berlin, Germany, April 1997. Springer-Verlag. http://www.uni-kl.de/AG-Nehmer/Ara/.

[28] Charles Perkins and David Johnson. Mobility Support in IPv6. In *Proceedings of the Second Annual International Conference on Mobile Coputing and Nettworking (MobiCom'96)*, Rye, New York, 1996.

[29] Christian Queinnec and Luc Moreau. Graceful Disconnection. In *Proceedings of International Workshop on Parallel and Distributed Computing for Symbolic and*

Irregular Applications, PDCSIA'99, Sendai, Japan, July 1999. World Scientific Publishing.

[30] Marc Shapiro, Peter Dickman, and David Plainfossé. SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection. Rapport de Recherche 1799, INRIA-Rocquencourt, November 1992.

[31] Gerard Tel and Friedemann Mattern. The Derivation of Distributed Termination Detection Algorithms from Garbage Collection Schemes. *ACM Transactions on Programming Languages and Systems*, 15(1):1–35, January 1993.

[32] Anand Tripathi, Neeran Karnik, Manish Vora, Tanvir Ahmed, and Ram D. Singh. Ajanta – A Mobile Agent Programming System. Technical Report TR98-016, Department of Computer Science, University of Minnesota, April 1999.

[33] Jan Vitek and Giuseppe Castagna. Seal: A Framework for Secure Mmobile Computations. In *Internet Programming Languages*, 1999.

[34] Mark Weiser. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, July 1993.