TRANSPARENT MIGRATION OF MOBILE AGENTS

Luc Moreau, Victor Tan and Nick Gibbins
Department of Electronics and Computer Science,
University of Southampton,
`L.Moreau,hktv99r,nmg97r@ecs.soton.ac.uk`

Abstract

SOFAR, the SOuthampton Framework for Agent Research, is used to build a pervasive information system, which provides context-adapted information to mobile users. Mobile agents are used as intermediaries between mobile users and the fixed infrastructure, migrating to locations with resources adapted to their objectives. SOFAR is equipped with an algorithm for tracking mobile agents and for transparently routing messages to them, without assuming the existence of any fixed or centralised control. In this paper, we present the algorithm and describe how it was integrated in the declarative style of SOFAR communications. Then, we present the agent meeting room: the realisation, in the agent framework, of a mechanism by which mobile users may exchange information during meetings.

1 Introduction

The target computing architecture for the coming years is that of computing devices carried by people and interacting with their environment using ad-hoc networking technologies. Many projects have started studying this environment, such as ubiquitous computing at Xerox [29], or wearable computing at MIT [25]. A common assumption is that mobile users may be connected to the network via wired or wireless means, and many maintain only intermittent connectivity with the network.

At Southampton, our aim is to design and build the architecture that will provide mobile users with the information they need, depending on their location, the time, or the presence of other mobile users in the environment. The *pervasive information fabric* [8] is the name we give to such an environment, which provides context-adapted information to mobile users.

The notion of agents is regarded as particularly suited for such a task. The AI community defines an *agent* as an entity situated in some environment and capable of flexible, autonomous actions in order to meet its design objectives [13]. Amongst others, the work of Maes [16] and Sycara [6] has popularised the idea that agents are particularly suited to provide users with customised information. We have coined the term "DIM Agent" to denote Distributed Information Management processes, which autonomously find, filter and present information adapted to the user [7, 21].

On the other hand, over the last few years, the notion of *mobile agent* has emerged as a new concept in the distributed computing community. A mobile agent is a running program that autonomously decides to change location in order to continue its execution in an environment with better resources. Software mobility is regarded as an evolution of the client-server paradigm [14], allowing computations to migrate to environments with resources more suited to their goals.

Our vision of mobile agents is that of itinerant and autonomous programs that execute their task in synergy with mobile users: a mobile agent is the representative of a mobile user operating in locations providing resources adapted to its objectives. Mobile agents start their existence on mobile devices, migrate to the infrastructure, interact with user-carried devices, and are able to return results to mobile devices when and if required by their specific goals.

Over the last two years, we have designed and built SOFAR, the SOuthampton Framework for Agent Research [21]. We regard this framework as the ubiquitous agent infrastructure coordinating the activities of information management agents. The design of SOFAR was influenced by ideas from the distributed computing, and in particular by the communication model of Nexus [11]. Not unlike the "*computational grid*" [10], which connects high-performance computing centers, we want to build an "*agent grid*", which connects information sources in the pervasive information fabric.

The goal of this paper is to describe the design of mobile agents in SOFAR. Our approach is characterised by the following salient features. *(i)* Communications are declarative and structured according to speech acts; there is a single interface for communicating with agents, whether fixed or mobile. *(ii)*

There are two ways of communicating with mobile agents. In the location-aware mode, messages are sent to a location; if the receiver has migrated, the communication fails. In the location-transparent mode, messages are routed transparently to mobile agents, wherever they are. *(iii)* A location tracking mechanism for mobile agents provides information used to route messages to them; the tracking and routing algorithms do not assume any fixed or centralised control, which makes our approach suitable to a pervasive computing environment. *(iv)* Agents operate on behalf of users, and SoFAR allows permissions to be granted to agents and users separately. *(v)* Ontologies have been developed for mobility-related concepts, in particularly to deal with meetings of users.

This paper is organised as follows. In Section 2, we introduce the fundamental ideas of SoFAR regarding the non-mobile facet of the framework. Then, in Section 3, we describe the algorithms for tracking agents and routing messages, and their integration in SoFAR. As agents act on behalf of users, we present in Section 4 how the association between agents and users is maintained in the framework. In Section 5, we describe the agent communication patterns that we use to build the meeting room [1], which is the mirror, in the agent world, of a meeting between users. Finally, we compare our contribution with related work, before concluding the paper.

2 SoFAR Overview

In this Section, we describe the three essential ideas that were adopted in SoFAR in order to make it a generic agent framework: *(i)* declarative communications, *(ii)* multiple communication modules based on the generic startpoint/endpoint approach, *(iii)* systematic use of ontologies. (Further details and discussion of the design can be found in [21].)

SoFAR does not prescribe the means through which communicative acts actually take place, but it does specify their intention. In SoFAR, communications are declarative and based on speech act theory [26]. A minimal set of *performatives* has been defined, representing the most common communication patterns, such as querying, informing, registering, subscribing or requesting. Other forms of communicative acts[1] must be defined by composing the primitive ones.

A speech act is associated with a *communication context*, which includes information about the act of communication itself such as the sender, receiver, time of sending, message identifier and conversation thread. This clearly differentiates our approach from distributed object systems, because an agent may use this information in order to reject messages, to discriminate between senders, or to determine which thread of conversation a message belongs to. We believe that *communication contexts* are vital information needed by agents in order to be able to exhibit autonomous behaviour.

During the design phase, we have integrated some of the ideas of the distributed computing community in the framework. In particular, ideas from Nexus [11], the communication layer of Globus and the "*computational grid*" [10] are at the heart of SoFAR.

A communication between two agents is based on a "virtual link" defined by a startpoint and an endpoint. An *endpoint* identifies an agent's ability to receive messages using a specific communication protocol; it extracts messages from the communication link and passes them on to the agent. A *startpoint* is the other end of the communication link, from which messages get sent to an endpoint. There may be several startpoints for a given agent, each acting as a representative (or proxy) of the agent at remote locations. Vice-versa, several endpoints may be associated with a startpoint, which allows some form of multicasting.

Practically, when an agent A_1 wishes to perform a speech act on another agent A_2, A_1 activates the corresponding method on a startpoint of A_2. The startpoint creates a communication context object, serialises the data, sends it to the endpoint, which deserialises the data and passes it to the agent A_2. When the speech act produces a result, it is returned back, via the endpoint, to the startpoint; it is then returned as the result of the method invocation on the startpoint. (Note that there is also an asynchronous variant of the performatives.)

The content of the messages sent between agents are terms in a simple knowledge representation language. These terms are implemented as serialisable Java objects, and are organised into ontologies

[1]We shall note that SoFAR uses speech acts like standard agent communication languages KQML and FIPA; however, SoFAR defines a minimal set of these, corresponding to the most frequent communication patterns.

(implemented as Java packages) which model particular problem domains.

3 Mobility Support

Mobile agents are programs able to migrate their code and state to new locations in order to make use of resources that are adapted to their task. Communicating with mobile agents is not a trivial problem because communication and migration are not atomic and may occur concurrently. SoFAR provides a location aware communication mechanism, which directs messages to fixed locations; if the recipient is no longer present, a failure occurs.

We have also designed another communication method that transparently routes messages to mobile agents; this mechanism hides the tedious details of tracking an agent and forwarding messages to it. In this Section, we first present the algorithm that we designed and then overview its implementation. In essence, the algorithm uses forwarding pointers combined both with a technique to collapse chains of pointers and with timestamps to prevent cyclic routing.

3.1 Algorithmic Foundations

Communications between SoFAR mobile agents rely on a new distributed algorithm that does not assume the existence of any fixed or centralised control. We believe that such an approach makes SoFAR an ideal candidate for an agent-based pervasive environment. The algorithm has been extensively studied in a previous publication [20], including its formalisation and its proof of correctness. We present here the essence of the algorithm and its integration in SoFAR.

Our assumption is that the algorithm is used at the application level (or more precisely at the agent level); we therefore require message delivery to be reliable. A mobile agent is associated with a timestamp, which is incremented at every migration. We use the term *platform* to refer to a host where mobile agents can migrate to and pursue their execution. Each platform maintains a table containing knowledge about mobile agents' positions and the timestamps they had when they were at these positions.

The algorithm has two components: a distributed directory service and a message router. Figure 1 describes the principles of the distributed directory service, which is responsible for maintaining information about the position of mobile agents in the system. There are three types of messages: AGENT, ACK and INFORM, respectively representing a migrating agent, the acknowledgement of its safe arrival, and location information propagation. In the text, we conventionally use numbers in bracket to refer to their correspondent in figures, where they identify successive messages and their respective effect on the local knowledge of a platform.

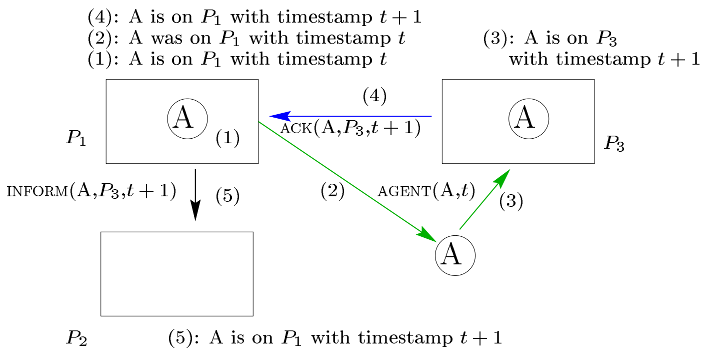
(4): A is on P_1 with timestamp $t+1$
(2): A was on P_1 with timestamp t
(1): A is on P_1 with timestamp t
(3): A is on P_3 with timestamp $t+1$
P_1
A
(1)
(4)
ACK(A,P_3,$t+1$)
A
P_3
INFORM(A,P_3,$t+1$)
(5)
(2)
AGENT(A,t)
(3)
A
P_2
(5): A is on P_1 with timestamp $t+1$

Figure 1: Distributed Directory Service

Initially (1), we consider an agent A, with timestamp t on platform P_1, which is aware of the agent's presence. When agent A decides to migrate, it informs its platform of its departure, which in turn updates its local knowledge (2); at this point, platform P_1 is only aware of the agent's departure, and is not required to know its destination yet. When agent A arrives at a new platform, the platform updates its local knowledge (3); note that the timestamp has now increased by one unit. In order to keep a trace to the agent, its previous location must informed by an acknowledgement message, specifying its location and timestamp (4). Timestamps are used to avoid race conditions between several acknowledgement messages arriving at a same platform; the receiver of a message updates its local knowledge if the local timestamp is smaller than the received timestamp. (Timestamps are an essential requirement of this algorithm because they prevent the formation of cycles in tables [20].)

This simple algorithm, combining agent migration, acknowledgement and timestamps, maintains enough information about the successive locations of mobile agents to be able to route messages to them; however, this approach leaves a trail of forwarding pointers that increases the cost of communication as agents migrate. Therefore, we use a further extra message, called INFORM, which propagates knowledge about agents locations between platforms. A range of strategies is available from lazy information propagation to eager broadcast. The latter is clearly not acceptable in a Internet wide context. In [20], we have established that it is safe to send such INFORM messages at any moment, from any platform, to any platform, because timestamps prevent cycle formation. In [23], we show that such INFORM messages may be piggy-backed on regular messages.

The message router is the component of the communication library that is responsible for reliably delivering messages between mobile agents. Its principle is summarised in Figure 2; it only deals with messages exchanged between mobile agents, and it utilises the local knowledge acquired by platforms using the distributed directory service.

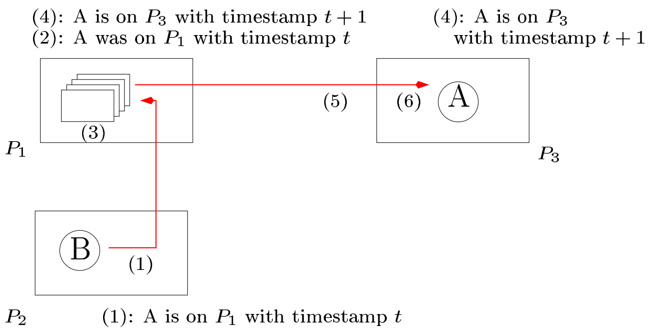
(4): A is on P_3 with timestamp $t+1$
(2): A was on P_1 with timestamp t
(4): A is on P_3 with timestamp $t+1$
(3)
(5)
(6)
A
P_1
P_3
B
(1)
P_2
(1): A is on P_1 with timestamp t

Figure 2: Message Routing

When an agent B wishes to send a message to agent A, it requests its platform to send the message on its behalf (1). Let us assume that platform P_2 knows that agent A was on P_1 when it had timestamp t, but in fact agent A has already left P_1 for another platform (2). The message is sent to platform P_1; as the agent is in transit, messages cannot be delivered to it, and therefore have to be accumulated (3). Once agent A reaches its new location P_3 and acknowledges its migration (4), messages accumulated on P_1 can be forwarded to P_3 (5). On the latter platform, agent A is known to be local and messages may be delivered (6).

In summary, platforms adopt the following algorithm. For any incoming message aimed at an agent A, the message will be delivered to A if A is known to be local. If A is in transit, the message will be enqueued, until the agent's location becomes known; otherwise, a message is forwarded to the agent's known location.

3.2 Implementation

As described in Section 2, SoFAR supports multiple communication mechanisms, all accessible through the startpoint/endpoint paradigm. Each communication module is identified by its name and must register its existence with a `Communication` class. New communication modules can be added dynamically, provided the permissions to do so have been granted. Each communication module must define three classes that respectively implement the interfaces startpoint, endpoint and boot (to be explained).

As a "server"[2], an agent chooses the communication mechanisms it wishes its clients to use. Communication mechanisms are simply selected by their name; currently, SoFAR supports Java Remote Method Invocation (RMI), shared memory (SHM), transparent routing for mobile agents (MOB) and secure socket layer based communications (SSL). For each of them, a triple containing a startpoint, an endpoint and a "boot object" must be returned by the corresponding communication module. Endpoints are kept private in an agent's implementation and are unforgeable. Startpoints may be exported and used by "clients". A "client" may choose the most suitable communication mechanism to interact with a server, among those selected by the server.

The boot object is used to bootstrap the communication and possibly to parameterise the protocol; for instance, with RMI, the bootstrap mechanism is the RMI-registry, and possible customisations involve the RMI port and machine.

The implementation of the transparent routing algorithm for mobile agents follows the same principle. A startpoint acts as a proxy to a mobile agent, and forwards speech acts to an endpoint. It is the endpoint's role to decide if a performative may be passed onto an agent locally, or if it should be forwarded. When an agent migrates, its associated endpoint becomes a message forwarder to a new endpoint created at the new agent's location. (Note that the old endpoint only becomes a forwarder after the new position has been acknowledged, as described in Section 3.1.) SoFAR relies on our implementation of *Mobile Objects in Java* [23], providing mobile endpoints, piggy backing of INFORM messages, and automatic clearing of routing tables.

In SoFAR, all agents are defined as a subclass of the `NullAgent` abstract class, providing, amongst others, a method to select communication mechanisms. Mobile agents are defined as a subclass of `MobileNullAgent`, which provides two methods `migrate` and `restart`. The method `migrate` initiates a migration to another platform, by requesting the current platform to transport the agent to the remote platform. As a result, the current agent state is serialised, passed to the remote platform, deserialised and reactivated by invoking the method `restart`. This form of migration is usually called *weak migration* [3].

3.3 Ontologies

Just as agents may have particular capabilities and characteristics, so too do platforms. A platform may have access to certain data resources by virtue of its network location, or be able to provide agents with an interface to a library of native code, or be located on a machine in a given physical location, or be run by a certain user. By reasoning about these platform capabilities, agents are given the ability to make informed decisions about which platforms they had best migrate to in order to achieve their objectives.

Each platform contains an agent that can describe the services provided by the platform, which are not necessarily the same as those offered by the agents present on the platform. The platform agent is owned by the user who started (and so owns) the platform.

We provide ontologies for a number of common domains, and these provide a foundation for describing platforms. (They are summarised in the left hand side of Figure 3; the right hand side will be explained in Section 5.2.)

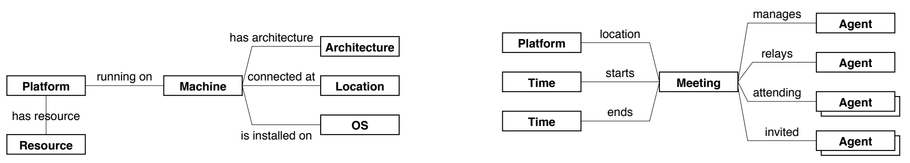

Figure 3: Ontologies for Platforms and for Meetings

[2]It is convenient to call the provider of a service a server and its user a client, even though one should not forget that agents act in a peer-to-peer manner.

4 User and Agent Identity

Agents operate on behalf of users, and this information must be managed by the framework so that permissions to perform “sensitive” operations may be granted to agents, depending on the users they are working for. In this Section, we present how SoFAR represents and manages users identities.

4.1 Agent Launching

The platform, which is a location where agents may be executed, is in practice a JVM that is able to process requests for starting up agents and for handling incoming mobile agents. Agents are started using an *agent launcher*. The agent launcher has access to an agent database, identifying agents, their classname, their codebase (i.e. the urls where their code can be downloaded from). Users identify the agent they want to start up, the platform where the agent must be activated, and optional arguments to be passed to the agent. The agent launcher sends a request to the platform, containing the user identity, the agent class name, its codebase and optional arguments.

When a platform receives such a request, it creates a new dynamic class loader, which loads the agent class from the specified codebase, and invokes its `main` method with the optional arguments. The launch is successful if the platform has the permission to start this agent from this codebase on behalf of this user. (The mechanism for granting permissions is explained below.)

When a platform receives a request for an incoming migrating agent, it reactivates its state, provided that this agent has the permission to be executed for its user, and of course, that the platform was able to load the code for the agent.

Note that agents may also be created by other agents: the created agent inherits from its parent the identity of the user that launched it.

4.2 Identity Awareness

SoFAR uses a certain number of mechanisms to ensure that agents do not have the ability to fake their identity or the identity of the user they work for. In this section, we present these mechanisms; they will be complemented by cryptographic techniques, which are currently under investigation, in order to authenticate (and possibly encrypt) the interlocutors of communication acts.

Granting Permissions to Users Using the JVM as the ubiquitous platform where agents can run causes interesting problems as far as user's identity is concerned. As agents perform actions on behalf of users, we would expect two instantiations of a same agent class to be given rights that may differ according to the user they are working for. It is therefore essential that an agent framework identifies the users who launch agents. Such a requirement is imposed by a range of applications, such as those that charge users for their service, or that manage the access to secure information; our system programmers had similar requirements because they want users to be accountable for their agents.

Unfortunately, the model of access control supported by Java does not take users into account. Stack inspection [28] is the mechanism that determines whether sensitive operations may be allowed. Permissions are allocated, via the policy manager, to classes' codebases and signatures. The stack inspection algorithm grants permissions to perform an operation if it is permitted by the codebase or the signature of all the methods that have an activation frame in the current stack. This is a low-level mechanism which offers very fine-grained control; we want to extend it to take users into account. Note that class signing is not a solution because it does not identify the user that is executing an agent, but the one who signed the code.

Our approach to this problem is to make sure that any code activated in SoFAR, sets up, at the bottom of the stack, a frame whose codebase uniquely identifies the user that is running the agent. By using the default Java policy management, we are able to grant permissions on a user-by-user basis. Such a user activation record has to be installed when a new thread is created, i.e. in three cases: *(i)* when an agent is launched, *(ii)* when an agent migrates, or *(iii)* when messages are received.

Platforms handle the first two cases. Following the request to launch an agent by the agent launcher, an agent is activated in a new thread, with a user-specific activation frame. Similarly, after receiving the request to `restart` a migrating agent, a platform executes its `restart` method after creating a user-specific activation frame. In both cases, the receiving platform has to trust the request's emitter to correctly identify the user that launched the agent. Authentication may be used to correctly identify platforms and users that launch agents, but clearly, proper trust management [4] has to be investigated.

Endpoints handle the third case: they receive messages, and activate their handler in a new thread after creating a user-specific activation record. As opposed to the first two cases, the user's identity is not propagated by the request emitter, but is given by the user that launched the receiving agent.

Agent Identity It is a requirement of SoFAR that all agents are defined as a subclass of the system class `NullAgent`. `NullAgent` is responsible for creating a read-only data structure that identifies an agent; it contains: the identity of the user that launched the agent, the agent's class, the JVM where it was activated, and a (locally) unique identifier.

Sender Identity Using the communication context, the recipient of a message can determine its sender. The framework must be able to prevent agents from faking the identity of a sender. This functionality is delegated to startpoints.

A startpoint acts as a proxy for an agent. A speech act with an agent is performed by calling the associated method on the agent's startpoint. The startpoint is responsible not only for communicating the argument of the speech act to the agent (via its endpoint), but also for defining the sender's identity.

A startpoint contains a transient field, which must be initialised in order to perform communications with its associated endpoint. The only way to initialise a startpoint is by using a protected method of `NullAgent` that initialises the startpoint with the agent's identity. This guarantees that if a communication is initiated by a startpoint, it necessarily contains the identity of the sender agent. (Note that an agent cannot request another agent to initialise a startpoint on its behalf, because communicating startpoints resets the transient field[3].)

Note that this mechanism does not prevent an attacker operating outside the framework from faking the identity of an agent of the framework. The solution to this problem is to use cryptographic techniques in order to authenticate the sender as a properly defined SoFAR agent.

5 Agents Communication Patterns

An agent framework must provide instantiable communication patterns [1] representing frequently used mobile agent interactions. The three patterns described in the next section can be used for any type of agents, and in particular for mobile agents that support transparent routing of messages. They are followed by an abstraction of the meeting room scenario.

5.1 Registration – Subscription – Relay

Every agent knows of one registry agent which it can use to advertise its capability, or to find out about the capabilities of other agents. *Registration* is the speech act by which an agent declares to the registry agent its capability to handle specific messages. If the registry answers positively to a registration act, it commits itself to advertise the registered capability and to return it to agents which ask matching queries.

As a proof of its commitment, the registry issues a *contract* as a result of the registration act. As long as the contract remains live, the registry will retain the advertised capability. Conversely, if the agent that registered the capability desires to stop its advertising, it just has to terminate the associated contract.

Subscription is the speech act by which an agent declares its interest in receiving specific messages. If a subscription manager answers positively to a subscription act, it commits itself to inform the subscribee

[3]Our assumption is that the communication layer uses serialisation, which resets transient fields. A communication that does not use serialisation would break this invariant. Therefore, initialisation of communication module is subject to a security check, for which a permission has to be granted by the platform.

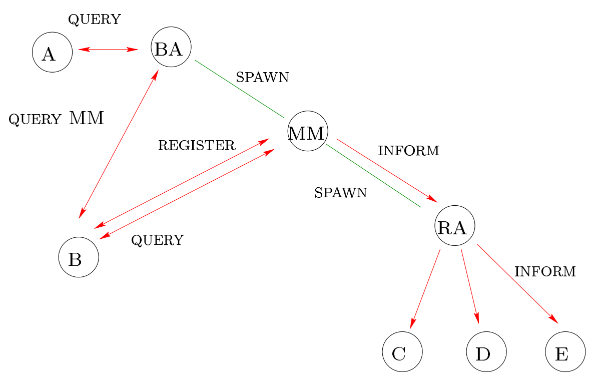

Figure 4: The Meeting Room Scenario

about facts that match its interest. As a proof of its commitment, the subscription manager also issues a *contract* as a result of the subscription act.

Contracts are first-class objects that may be shared and exchanged by agents. They are used to detect the termination [27], between distributed agents, of the registration or subscription procedures.

While the subscription manager is used to selectively multicast facts to a set of agents, depending on their preferences, the relay agent blindly multicasts information to a set of member agents. With the relay agent, an agent does not declare its interest by a subscription act, but instead members must be handled by a third agent. The relay agent is able to propagate queries to all the members, and collate and return results back to the querant: it therefore offers a pattern for querying groups of agents.

5.2 The Meeting Room

Our motivation is to build an agent infrastructure for the pervasive information fabric [8], the ubiquitous environment that brings to the user the information they need as a function of their location, the time, or the presence of other mobile users. A particular example is the meeting room scenario, for which we have prototyped an implementation in SoFAR.

Imagine you arrive at a meeting. Using the local ad-hoc networking infrastructure, the computing device you carry starts interacting with those of your counterparts in the room. Documents related to the meeting are exchanged by the devices, e.g. minutes of the previous meeting, but also personal annotations on documents, remarks. These documents and cross-references, i.e. hyper-links, constitute a hyperweb, which was created by the presence of some users in the meeting room, and which can be browsed by each user.

The scenario is an illustration of the pervasive information fabric that shows how “ad-hoc hyperdocuments” may be created. In this paper, we do not focus on the creation of links between documents, which is the subject of another publication.

As we regard mobile users and mobile agents as complementary, our route to build the meeting room scenario is to create a virtual meeting room for mobile agents. Our implementation builds upon the basic premise of a meeting as presented in [1], and extends it to incorporate other concepts which mirror typical real-life environments. In particular, we introduce the idea of public and private meetings, the constraints of starting and ending times for a meeting, and finally the idea of associating agents directly with users. A public meeting can be attended by any agents aware of its existence, whereas a private meeting can only be attended by “invited” agents. We have implemented an ontology (right-hand side of Figure 3) which allows agents to communicate about these aspects of a meeting.

For a private meeting, an agent (A in Figure 4) has to make a request to a booking agent (a permanent agent on all platforms in the system) to set up a meeting, providing the list of agents that are “invited” to attend. Assuming agent B is invited, it then migrates over to the platform where the meeting is held, where its presence is validated. The booking agent then spawns the meeting manager, with whom agent B registers itself, receiving a contract in return. Subsequent arriving agents are validated and registered in a similar manner. Invited agents may query the meeting manager to obtain the list of participating agents that have arrived so far.

The meeting commences either when the starting time is reached (weak attendancy), or when all invited agents have arrived and registered with the meeting manager (strong attendancy). In either case, a relay agent is spawned and its identity distributed to all participants. Agent interactions then commence; each agent may send a message to the relay agent, which in turn broadcasts it to all other agents in the meeting.

The meeting will terminate either when the end time is reached (this event will be broadcast to all agents), or when the attending agents leave the meeting. The meeting manager and relay agent will both shutdown, cancelling all contracts and releasing all system resources still used.

A public meeting negates the need for attendance lists and validation; the booking agent merely advertises meetings on specific topics via a system registry. This registry is accessible to a local community of agents who can query it, match registered topics with their own interests, and decide which platform to migrate to in order to participate. Public meetings enforce a predefined starting and ending time.

A facility is provided for specifying that an agent representing a specific user should attend a particular meeting. An invitation is directed to an agent that represents a user, which can then (with or without user intervention) either relay the invitation onwards to some previously spawned agent, or spawn a new agent to attend the meeting.

In SoFAR, we have the ability to simulate meetings behind closed doors. First, if an attendee migrates out of the platform, its contract is immediately terminated (during the serialisation process); therefore, it no longer receives messages from the other participants. Furthermore, it is possible to forbid attendees from communicating with agents on other platforms by preventing them from initialising startpoints.

6 Related Work

There is an amazing number of mobile agent systems, and we cannot list them all in this paper. We refer the reader to WWW-sites such as [24, 12]. Our literature survey indicates that no other agent system combines the key features of SoFAR: *(i)* declarative communications based on speech acts and independent of the transport layer; *(ii)* communication layer that routes messages transparently to mobile agents and does not rely on any fixed or centralised locations; *(iii)* propagated identities of users and agents so that permissions can be granted to agents and users independently.

A number of systems such as GrassHopper or Voyager support multiple communication modules; others, such as Hive [18] or Ajanta, use RMI only. We note that adopting the startpoint-endpoint paradigm abstracts away from the communication details that sometimes make programming communications very tedious. It also elegantly helps us encapsulate the management of agent identity or communication encryption.

We agree with Wojciechowski and Sewell that "*A wide-area programming language should provide a level of abstraction that makes distribution and network communication clear; higher levels should be provide and implemented using the modularisation facilities of the language.*" [30]. Our modular approach to communications follows the same principle, with the choice between straight point-to-point communication or transparent routing made by the agent implementor, or even dynamically by the agent itself.

We observe that there are relatively few agent systems that offer transparent point-to-point communications with mobile agents. This is problematic because, in the interval between finding an agent and initiating communications with it, the agent may have migrated, causing a communications failure. Voyager provides a "redirecting proxy", which is able to redirect a communication to the next agent location. We are not aware of the details of the algorithm used, but our investigation [20] shows that naive message forwarding and race conditions may result in cyclic routing when an agent visits a previously visited site: it is the reason why our algorithm introduces timestamps.

Nomadic Pict [30] uses several location-independent algorithms, implemented by converting location-independent primitives into a core calculus with explicit locations; the algorithm they present uses a fixed query server. A fixed component is also adopted by April [17] and JumpingBeans [2]. As we regard SoFAR as an agent infrastructure for the pervasive information fabric, we believe that adopting fixed or centralised locations would place an excessive burden on the infrastructure.

Other mobile agents systems opt for completely different communication mechanisms. Mars [5] follows coordination-based approaches and Mole [3] has an event model that supports mobile participants. Some of this functionality is provided in our framework by the subscription and relay mechanisms, combined with transparent message routing.

Scalability of mobile agent systems has different facets including efficiency of message delivery for roaming agents, size of routing tables and efficiency of brokering. We are investigating these issues, by exploring different avenues. The problem of clearing routing tables is equivalent to the distributed termination problem [27, 20], and therefore distributed reference counting techniques may be used for that purpose. Hierarchical organisations, as noted by [15], help dealing with many components in a system [19]. Techniques such as query routing [9] are also applicable in this context, and they should be compared with "flooding" techniques.

7 Conclusion

The SoFAR framework has been the focus of a tremendous activity involving up to thirty researchers in the *Intelligence, Agents and Multimedia* group at Southampton. Training sessions were organised about agents, ontologies, and the actual framework implementation in Java. On two occasions, a group activity, called "agentfest", took place: during a three day-session, those researchers developed agents, operating on application domains such as: user interfaces, image processing, audio and video segmentation, conversion between audio and text, cooperative applications, and linking services. As a result, SoFAR has now been adopted by several researchers for their every day research.

The startpoint and endpoint abstraction, combined with transparent routing of messages, are very powerful techniques. In particular, they considerably simplify the implementation of groups of mobile agents [22]. These groups and the associated management of resources will be included in the framework.

The goal of this agent framework is to build the *pervasive information fabric*, and we are using the meeting room so that agents may exchange link databases and find information that is relevant to their users [8]. In that context, security becomes an essential issue: we have identified the need for authentication and trust management, which we are now investigating.

8 Acknowledgement

This research benefited from discussions with colleagues in the *Intelligence, Agents and Multimedia* Group at Southampton, including David DeRoure, Tim Chown, Mark Thompson, Kevin Page. In particular, David De Roure coined the term "pervasive information fabric". Thanks to the numerous researchers who have used SoFAR.

References

[1] Yariv Aridor and Danny B. Lange. Agent Design Patterns: Elements of Agent Application Design. In *Proceeding of Autonomous Agents (AA'98)*, 1998.

[2] Ad Astra. Jumping beans. Technical report, White Paper, 1999. http://www.JumpingBeans.com/.

[3] J. Baumann, F. Hohl, K. Rothermel, and M. Straer. Mole - Concepts of a Mobile Agent System. In 3, editor, *World Wide Web*, volume 1, pages 123–137, 1998.

[4] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The Role of Trust Management in Distributed Systems Security. In Jan Vitek; Christian Jensen, editor, *Secure Internet Programming*, number 1603 in LNCS, pages 185–210, 1999.

[5] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination . In *Proceedings of the 2nd International Workshop on Mobile Agents (MA'98)*, number 1477 in LNCS, 1998.

[6] L. Chen and K. Sycara. WebMate: a Personal Agent for Browsing and Searching. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 132–139, 1998.

[7] David DeRoure, Wendy Hall, Hugh Davis, and Jonathan Dale. Agents for distributed multimedia information management. In *Pratical Application of Intelligent Agents and Multi-Agent Systems*, pages 91–102, London, UK, April 1996.

[8] David DeRoure, Danius Michaelides, and Mark Thompson. Weaving the Pervasive Computing Fabric. Submitted for publication, January 2000.

[9] David C. DeRoure, Samhaa El-Beltagy, Nicholas M. Gibbins, Les A. Carr, and Wendy Hall. Integrating link resolution services using query routing. In *5th Workshop on Open Hypermedia Systems (OHS5)*, Darmstadt, Germany, February 1999.

[10] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman Publishers, 1998.

[11] Ian Foster, Carl Kesselman, and Steven Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing*, 37:70–82, 1996.

[12] Fritz Hohl. Security in Mobile Agent Systems. `http://mole.informatik.uni-stuttgart.de/security.html`.

[13] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 2000.

[14] Danny B. Lange and Mitsuru Ishima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[15] Sashi Lazar, Ishan Weerakoon, and Deepinder Sidhu. A Scalable Location Tracking and Message Delivery Scheme for Mobile Agents. Technical report, University of Maryland, 1998.

[16] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[17] 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.

[18] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krilorian, and Pattie Maes. Hive: Distributed Agents for Networking Things. In *The joint Symposym ASA/MA 1999*, 1999.

[19] Luc Moreau. Hierarchical Distributed Reference Counting. In *Proceedings of the First ACM SIGPLAN International Symposium on Memory Management (ISMM'98)*, pages 57–67, Vancouver, BC, Canada, October 1998. Also in *ACM SIGPLAN Notices*, 34(3):57–67, March 1999.

[20] Luc Moreau. Distributed Directory Service and Message Router for Mobile Agents. Technical Report ECSTR M99/3, University of Southampton, 1999.

[21] 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. SoFAR with DIM Agents: An Agent Framework for Distributed Information Management. In *The Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 369–388, Manchester, UK, April 2000.

[22] Luc Moreau and Christian Queinnec. Distributed Computations Driven by Resource Consumption. In *IEEE International Conference on Computer Languages (ICCL'98)*, pages 68–77, Chicago, Illinois, May 1998.

[23] Luc Moreau and Daniel Ribbens. Mobile Objects in Java. Technical report, University of Southampton, 2000.

[24] Reticular Systems, Inc. AgentBuilder. `http://www.agentbuilder.com/`.

[25] Bradley J. Rhodes. The Wearable Remembrance Agent: A System for Augmented Memory. *Personal Technologies Journal*, 1(4):218–224, 1997.

[26] John Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[27] 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.

[28] Dan Seth Wallach. *A new Approach to Mobile Code Security*. PhD thesis, Priceton University, 1999.

[29] Mark Weiser. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, 36(7):74–84, July 1993.

[30] Pawel Wojciechowski and Peter Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. In *First International Symposium on Agent Systems and Applications/Third International Symposium on Mobile Agents (ASA/MA'99)*, October 1999.