Agent Management Support for Mobility

Proposal version 1.2—6th April, 1998
Jonathan Dale and Francis G. McCabe
Fujitsu Laboratories of America, Inc.
595 Lawrence Expressway, Sunnyvale, California, 95086, USA
jdale@fla.fujitsu.com, fgm@fla.fujitsu.com

1. Terms and Definitions

The nomenclature used within this proposal has been extended from the terms given in the Common Conceptual Model of the Mobile Agent Facility (MAF) specification [8]. Briefly:

- An agent is software that acts autonomously on behalf of another entity.
- A node is an addressable computer that is connected to a network.
- A stationary agent is an agent that executes only upon the node where it begins executing and is reliant upon it.
- A mobile agent is an agent that is not reliant upon the node where it began executing and can subsequently transport itself between nodes.
- An agent name uniquely identifies an agent within the network. Stationary and mobile agents communicate chiefly through agent names.
- Mobility is the property or characteristic of an agent that allows it to travel between nodes.
- An agent system is a platform that can execute both stationary and mobile agents. Multiple agent systems can reside on a node.
- Agent migration is the process by which an agent transports all or part of itself between agent systems. Agent migration is different from process migration in that it involves a higher level of negotiation and cooperation between mobile agents and agent systems.
- Agent cloning is the process by which an agent creates a copy of itself, identical in every respect except for their agent names.
- Agent invocation is the process by which an agent can invoke other agents on remote agent systems. The act of agent invocation is distinct from agent migration in that the invoking agent can be totally unrelated to the agent being invoked; in agent migration and agent cloning, some relation of identity is usually implied.
- Agent state describes the code and state of the agent. A mobile agent can transfer all or parts of its agent state when it migrates between agent systems.

2. Introduction

Mobility is a characteristic that can be applied to software which allows it to move between machines. However, since this in itself is such a broad definition, it is important to note that the movement of programs is fundamentally different to the movement of executing software; the former being primarily a data transfer operation and the latter involving capturing some notion of the state of the software.

The transfer of executing software between machines has already been investigated by distributed operating systems research and is termed process migration [13]. This operation typically involves suspending a process at a given point, marshalling its program context\(^1\) and transferring it to another

---

1. In a particularly process-oriented view, a program context consists of the code, data and stack segments of a process. In a more general view, a program context can consist of various configurations of these segments.
machine and then restarting the process at the point after which it was stopped. In process migration, it is usually the underlying distributed operating system that decides when and where a process should be migrated as part of its management operation, mainly with a view to improving load balancing, fault tolerance and parallelism. Additionally, because process migration is such a low-level operation it is normally confined to architectures and software configurations of the same order.

Mobility as a characteristic of agents is a field that can be said to be growing from two directions; as a distributed communication model and also as a distributed computation model. As a communication model, mobility allows an agent to move across a network to communicate locally with a service provider or resource. As a computation model, mobility allows agents to move across a network to execute at the most appropriate location specified by their requirements.

However, although mobile agency is seen as a controversial subject, mainly due to security, resource access and control issues and lack of perceived need, it can be a more natural solution for certain types of problems, especially those involving distribution in large-scale networks such as the Internet. One potential application area of mobile agents is in helping to support users on mobile computers, for example:

A user is working on their portable computer. They decide on a number of network-based activities and specify their goals and parameters of operation. They connect their machine to the network at which point these activities are embodied as executing agents and move from the environment of the portable computer on to a node which is permanently connected to the network. The user is now free to disconnect from the network and continue working in an off-line mode. During this time, the user’s mobile agents are moving between nodes and performing their given tasks. Periodically, the user reconnects to the network to check on the status of their agents and possibly to launch more agents or to reconfigure previously executed agents.

As will be shown later, the small number of primitives that are required to support mobility in agents can also support other related functions, such as remote agent configuration and agent cloning. It also provides great flexibility in describing various protocols for agent mobility and allows an agent to direct the control of its mobility operation in the event of unexpected circumstances or problems.

This document describes a proposal to support agent mobility within the FIPA 98 specification that is based upon a flexible and extensible form of mobility. Initially, the requirements and life-cycle of mobile agents are detailed as well as a number of mobile agent-related issues. Next, the minimum set of primitives and protocols needed to facilitate various forms of mobility are presented; the protocols are written in terms of the identified primitives and describe the necessary interactions with the agent systems involved. The way in which the primitives and protocols address the highlighted mobile agent issues is then discussed. A mechanism follows, called a language-binding, for abstracting the functionality and details of the underlying programming language or mobile agent system. The document ends with a critique of this approach and an appendix in which the primitives are described in terms of the FIPA agent management ontology.

3. Requirements for Mobility

The user-based notion of mobility is centred around the projection of some aspect of an agent’s behaviour. In terms of some existing agent-based application areas, these can be expressed as:

- **Personal assistants** which deal primarily with projecting various personal attributes of the user’s self, such as diary information or user interests.

- **Information management** which deals primarily with projecting various control attributes of the user’s information, such as document access and modification rights.

- **Electronic commerce** which deals primarily with projecting various authority attributes of the user’s requirements, such as purchasing information and authorisation.
• **Distributed systems modelling** which deals primarily with projecting various execution attributes of the user’s distributed computation, such as parallelisation and location.

To a large extent, depending upon the domain of applicability of a mobile agent, these types of projection overlap and intersect. For example, a personal assistant may also have to manage a user’s information actively to ensure its integrity.

Similarly, at a programming level mobility refers to the projection of the agent, or some portion of the agent, on to a node in the network:

• **Agent migration** where an agent moves itself or part of itself to another node.
• **Agent cloning** where an agent creates a duplicate of itself.
• **Agent invocation** where an agent invokes another agent.

Each of these operations involves some transfer of the state of an agent and also a transfer of identity. With agent migration and cloning, the identity of the agent before the operation is the same or similar to the identity of the resultant agent. However, with agent invocation the identities of the two agents are distinct. This makes agent invocation suitable for remote agent configuration, where a solution involving multiple agents can be preconfigured to execute across a networked environment.

From these three perspectives, it is possible to derive a set of requirements that mobile agents require in order to function correctly within an agent system:

• **Mobility**—The ability to move an agent. The movement details involved in agent migration are more complex than those required for process migration. Agent-level migration requires marshalling, transference and negotiation between the agent and the agent system involved; process-level migration simply requires marshalling and transference. Agent migration can also require other mechanisms to be in place such as checkpointing and message loss prevention for fault tolerance and message redirection for migration, if these are so required.

• **Management**—The ability to control an agent. For an agent system to be able to execute a mobile agent, it must also have the necessary mechanisms to allow it to control its operation. Agents from untrusted (and even trusted) agent systems could potentially affect the host agent system or node in some adverse way. The form of this adversity could be simply poor programming which inefficiently utilises some system resource, or could be malicious in that it attempts to control or monopolise some system resource. Both of these situations could affect the performance of the agent system as a whole (to its users and agents) and could also compromise the security of the data contained within it. In this way, mobile agents have to be managed with regard to the resources that they request.

• **Security**—The ability to empower or restrict an agent. Mobile agents provide particular problems with regard to maintaining the integrity of an agent system and its data, as mentioned above, but they also pose a potential security threat. Since it is typically code (of one form or another) that is being transferred, the security aspect of the agent system needs to make sure that the code is of a suitable form to execute. Whereas management is concerned with controlling access to system-level resources, such as processor cycles and memory usage, security is concerned with controlling access to application-level resources such as the agent system, agents and information resources. This means that security involves such concepts as user access rights, authentication, verification and authorisation.

• **Independence**—The ability to move regardless of programming language. The underlying techniques for mobility (agent migration, agent cloning and agent invocation) should not be tailored to either a specific programming language or mobile agent system. However, the proposal recognises the existing body of work that has taken place in this area and makes provision for this by introducing language bindings later in the document.
One of the major underlying requirements for mobile agents is the ability to communicate not only with other agents but also with other agent systems. In this respect, mobile agents also require a communication model that is based upon:

- **Asynchronous communication.** The ability for agents to interleave many different communications.
- **Peer-to-peer conversations.** The ability for agents to act as clients and servers according to need.
- **Message management.** The ability for agents to send messages transparently.
- **Interoperability.** The ability for agents to converse by using the same protocol.
- **Ontological consistency.** The ability for agents to be understood by using the same vocabulary.

These and various other aspects are already addressed in the FIPA 97 specification [4] or are to be addressed in the next revision of the specification [5]:

- **Part 1: Agent Management** covers basic agent component specification, management and authentication.
- **Part 2: Agent Communication Language** covers synchronous communication between agents, but it needs to be augmented to support an asynchronous communication model (cf. 7.1: Asynchronous Messaging proposal [10]).
- **Part 10: Agent Security** will cover security issues for stationary and mobile agents.
- **Part 12: Ontology Service** will cover ontological issues for all agent specifications.

4. **Life Cycle of a Mobile Agent**

4.1. **Agent States and Transitions**

In most current mobile agent system literature, the definition for agent mobility invariably equates to that of agent migration. However, mobility is a **concept** that can be applied to agency whereas migration is a **technique** that can be used to achieve a certain form of mobility: it is important to maintain this distinction if flexible agent mobility is to be supported that is independent of implementation. By this token, agent mobility encompasses more than just agent migration because it allows other forms of agent movement to be expressed, such as subprogramming and remote invocation.

The operation of a mobile agent is similar but subtly different from that of a stationary agent: a mobile agent need not execute all of its instructions on one node. This has ramifications for the life cycle of an agent, since an agent can now be in one of five distinct states (see figure 1 which has been modified from the agent life cycle diagram given in part 1 of the FIPA 97 specification):

- **Initiated.** The agent is created.
- **Active.** The agent is currently executing.
- **Suspended.** The agent has been suspended.
- **Waiting.** The agent is waiting on some event.
- **Transit.** The agent is in the process of moving.

Only mobile agents can enter the transit state, or to put it another way, stationary agents never enter the transit state. This ensures that a stationary agent executes all of its instructions on the node where it was invoked. The actions of agents can be described as (figure 1):

- **Create.** The creation (installation) of a new agent.
- **Invoke.** The invocation of a new agent.
- **Destroy.** The forceful termination of an agent. This can only be initiated by the agent system and cannot be ignored by the agent.
- **Quit.** The graceful termination of an agent. This can be ignored by the agent.
• **Suspend.** Puts an agent in a suspended state. This can be initiated by the agent or the agent system.

• **Resume.** Brings the agent from a suspended state. This can only be initiated by the agent system.

• **Wait.** Puts the agent in a waiting state. This can only be initiated by the agent.

• **Wake Up.** Brings the agent from a waiting state. This can only be initiated by the agent system.

• **Move.** Puts the agent in a transitory state. This can only be initiated by the agent.

• **Execute.** Brings the agent from a transitory state. This can only be initiated by the agent system.

Again, the last two actions are only used by mobile agents.

### 4.2. Mobile Agent Issues

Although the life cycle illustrates the main states and transitions available to an agent, there are some support issues that need to be addressed. These particularly concern the movement of a mobile agent to and from the *Transit* state and also how it is handled whilst in this state. These issues can be summarised as:

• **Agent transfer**—How is a mobile agent to be moved between agent systems?
  After a mobile agent has initiated transfer, the agent needs to be sure that the migration process has taken place successfully before it terminates its old incarnation and moves over to its new incarnation. Also, is the mobile agent an active or a passive entity in the migration process? If it is passive, then the transfer of the mobile agent to an agent system is transparent; if it is active, then the agent is aware of every stage of the process.

• **Message loss prevention**—How are messages to be handled to prevent loss?
  When a mobile agent moves between agent systems, its network name will change. Messages that have been sent to the old network address need to be handled to ensure that a mobile agent does not miss any messages either when it is in transit or after it has moved. This could be achieved by placing redirection pointers within each agent system that the mobile agent has visited, but this can lead to chains of pointers being created across agent systems. Additionally, for
intermittently connected devices, messages need to be queue by another agent system while the
device is not connected to the network.

- **Agent loss detection**—How is it possible to tell if an agent has been lost?
  Since mobile agents do not need to rely on the node and agent system that launched them, this
  makes monitoring an agent's progress more difficult. Agent monitoring can be used to track the
  status of an agent and can also be used to determine if a particular agent has been lost either in
  transit or through failure on a particular node or agent system. Moreover, checkpointing a mobile
  agent at every stage of its movement can allow a user to roll back to a previous version of the
  agent if a future version becomes lost.

- **Agent management**—How is an active agent to be controlled?
  In environments where untrusted agents (whether mobile or stationary) are being executed,
  safeguards need to be in place to protect the integrity of the agent system and node, and also to
  protect the security and secrecy of the data stored there. Current mobile agent research literature
  (for example [14, 6, 11]) has proposed that agents can be controlled through their access to
  resources; application-level resources such as service access, and system-level resources such
  as processor cycles and memory. This can not only help to prevent agents performing prohibited
  actions (security) but can also help to ensure that agents cannot swamp a node or monopolise a
  resource (management).

5. **Mobility Management Primitives**

5.1. **Primitive Rationale**

As mentioned in section 3, the following forms of mobility are proposed to be supported in the FIPA 98
specification:

- **Agent migration**. The movement of an agent from one agent system to another by marshalling,
  transferring and executing all or part of a mobile agent.

- **Agent cloning**. The duplication of an agent by marshalling and executing all of a mobile agent.

- **Agent invocation**. The creation of a new agent by marshalling, transferring and executing the code
  of one agent by another agent.

These forms of mobility exist within a space that can be defined by three dimensions:

1. **Type**—All, Part or Other.
   The type value determines what part of the agent is to be considered for moving; either all of the
   current agent, part of the current agent or all of another agent.

2. **Location**—Local or Remote.
   The location value determines where the agent is to move; either on the local agent system or to a
   remote agent system. Moving to a remote agent system means both a different agent system on
   the local node or to a remote node.

3. **Original**—Survives or Terminated.
   The original value determines what happens to the agent which is requesting the move after the
   operation completes; either the agent survives or it is terminated.

Existing mobility operations that are found in current mobile agent systems can be represented by
combining various configurations of these values, for example:

- **Type=All, Location=Remote, Original=Terminated** is the Telescript 'go' operation.
- **Type=All, Location=Local, Original=Survive** is a traditional 'clone' operation.
- **Type=Partial, Location=Remote, Original=Survive** is a remote ‘fork’ operation.

*Language bindings* help to keep each form of mobility independent from its implementation and also
allow the specific features of either the underlying programming language or the mobile agent system to
be expressed. Section 7 talks more about the form of language bindings, but for the present, a language binding is considered to be a data object that contains a number of attribute-value pairs:

- A block of data representing the agent state of the mobile agent being transferred.
- An identifier describing the nature of the block of data.
- A set of attribute-value pairs describing other features of the mobile agent, which are perhaps specific to a particular programming language or mobile agent system.

If an agent system supports a particular programming language or mobile agent system, then it will also know how to execute a code segment (or how to have it executed) and how to interpret the specified features. In this way, agent systems can use this information to determine if they can accept and execute a mobile agent before it is physically moved.

5.2. Primitive Description

Table 1 gives a summary of the proposed primitives which are required to support agent mobility. In this model, mobility is split into two distinct operations; the Move primitive which initiates the movement of a mobile agent and the Transfer primitive which completes the process by transferring the identity and authority of an agent. Fuller forms of the primitives in the style of the FIPA 97 specification are given at the end of this document in Appendix A.

Both the Move and Transfer primitives are modelled as FIPA Agent Management Request performatives [4] that are sent to an agent system. Figure 2 gives an example of the Move primitive wrapped in a Request performative in which a mobile agent (some-mobile-agent) is asking for its agent state (in the form of language-binding) to be transferred across to another agent system (some-agent-system). The mobile agent has also specified a list of resource requirements that it needs; if these requirements cannot be met or the agent system cannot process this form of language binding, then the movement operation will fail.

The complement of Move is the Transfer primitive which is used to complete the movement operation by conferring an identity and authority on the newly created agent. In figure 3, a new agent has been created (some-new-agent) which is being granted a certain identity and authority that is contained within the signature that is being transferred.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>Moves an agent or a copy of an agent to an agent system.</td>
</tr>
<tr>
<td>Transfer</td>
<td>Transfers the identity (and authority) of an agent to another agent.</td>
</tr>
<tr>
<td>Execute</td>
<td>Executes an agent on an agent system.</td>
</tr>
<tr>
<td>Quit</td>
<td>Gracefully terminates the execution of an agent.</td>
</tr>
</tbody>
</table>

Table 1 : Primitives Required to Facilitate Agent Mobility
6. Mobility Protocols

The next subsections show how the primitives can be combined into protocols to support the three forms of mobility identified earlier. However, it should be noted that these protocols are here to illustrate example use of the primitives only, because a problem with deciding how to use the primitives is in determining between mechanism and policy. These protocols have been designed to minimise policy by embedding migration control within the mobile agent. This gives it the greatest flexibility in deciding how to migrate and how to control migration in the event of something unexpected.

Other protocols are entirely possible with these primitives which embedded migration control within the agent systems and also make migration appear transparent to the agent, as per Telescript and others.

6.1. A Protocol for Agent Migration

Migration is the process of moving something from one locality to another; in mobile agency, agent migration refers to the relocation of a mobile agent from within one agent system to within another agent system. Agent migration is the primary mechanism by which agents can become mobile. The form of migration, however, will depend largely upon the programming language or mobile agent system that the mobile agent was written in, which is specified as part of the language binding (see section 7).

(request
  :sender some-mobile-agent
  :receiver some-agent-system
  :content
  (action some-agent-system
   (move
    (:agent-name some-mobile-agent)
    (:agent-requirements ...)
    (:language-binding ...)n
    (:ownership owner-name))
  :protocol fipa-request
  :ontology fipa-agent-management)

Figure 2: Example Move Request

(request
  :sender some-mobile-agent
  :receiver some-agent-system
  :content
  (action some-agent-system
   (transfer
    (:agent-name some-new-agent)
    (:signature ...)n
    (:ownership owner-name))
  :protocol fipa-request
  :ontology fipa-agent-management)

Figure 3: Example Transfer Request
An example protocol for agent migration is given in figure 4:

1. Agent A has decided to migrate itself from the Local Agent System (which is the agent system upon which it is currently active) to a Remote Agent System (which may or may not be hosted on a remote node) and issues a Move request to that intended Remote Agent System. This request contains a suitable language binding and other information pertinent to the migration, such as Agent A’s credentials and resource requirements.

2. The receiving Remote Agent System processes the request and determines whether Agent A can be accommodated according to its features. If it cannot, then nothing happens; Agent A times out and the current migration operation is aborted. If it can, then the Remote Agent System arranges to have the code of Agent A run by issuing an Execute request. This results in another agent being created, called Agent A’, which exists in a limited state.

3. Agent A’ determines if it has been migrated successfully and informs Agent A. If it has not, then Agent A terminates Agent A’ and the current migration operation is aborted. An example of unsuccessful migration could be where the resources requested by Agent A become unavailable during movement and subsequent execution.

4. If Agent A’ has been migrated successfully, then Agent A informs its Home Agent System of its location change. This is necessary since it allows the owner of the mobile agent to keep track of the current location of all of its mobile agents.

5. The Home Agent System may accept or refuse this location change according to features of Agent A (such as its credentials) or user restriction (such as the user not wanting an agent migrated to that Remote Agent System). In such a case, Agent A terminates Agent A’ and the current migration operation is aborted.

6. Agent A also requests a change of location with the Local Agent System. This is necessary since it allows the Local Agent System to buffer any outstanding messages for Agent A and to redirect them to its new location.
7. The Local Agent System may also accept or refuse this location change request, according to features of Agent A (such as its credentials). In such a case, Agent A may decide to continue with migration with the knowledge that pending and future messages at the Local Agent System will not be redirected.

8. Now that all aspects of the migration operation have been dealt with, Agent A issues a Transfer request to Agent A'. This will install in Agent A' the same identity and authority as Agent A possessed. Agent A' now continues executing normally.

9. With a successful identity transfer, Agent A can terminate.

What is not shown in this protocol is Agent A having to unregister itself from the Local Agent System and also Agent A' having to register itself with both its Home Agent System and also with the agent system where it has migrated. These processes are explained in the Agent Management section of the FIPA 97 specification.

The order of each of these operations is crucial if Agent A is to be given the opportunity to direct the migration if an error occurs. The placement of control within Agent A gives it a great deal of flexibility in deciding what is actually part of the migration process and it can dynamically augment the protocol as it sees fit. Thus, a migrating agent can be assured that at the end of the migration process, it is safe to transfer its identity into its new incarnation and to terminate its old incarnation.

Agent migration is necessary to promote an agent culture of itinerancy [1], that is, the idea that mobile agents decide which nodes to visit based upon their goals and requirements and the features of specific agent systems. Itinerancy also helps to mobile agents to be autonomous and allows users to access their resources through indirect management techniques. Agent migration is being examined as a tool for information retrieval and data mining [12, 2], for electronic commerce [14, 7] and also as a tool for distributed information management, where mobile agents provide a management infrastructure in which users can publish, edit and browse information that is available across distributed resources [2]. All of these applications are particularly well-suited to large-scale networked environments, such as the Internet.

6.2. A Protocol for Agent Cloning

Cloning is the operation of creating a copy of something; in mobile agency, agent cloning refers to the creation of a duplicate mobile agent from an existing mobile agent. In essence, agent cloning is very similar to agent migration except that the identity and execution status of the original agent is preserved, and the original and the cloned agent continue execution in parallel. Additionally, since a cloned agent is an agent in its own right, it does not have exactly the same identity as its original and therefore does not require message redirection from the local agent system. The form of cloning, however, will depend largely upon the programming language or mobile agent system that the mobile agent was written in, which is specified as part of the language binding (see section 7).
An example protocol for agent cloning is given in figure 5:

1. Agent A has decided to clone a copy of itself on an Agent System (which could be a local or remote agent system) and issues a **Move** request to that Agent System. This request contains a suitable language binding and other information pertinent to the cloning operation, such as Agent A’s credentials and resource requirements.

2. The receiving Agent System processes the request and determines whether the clone of Agent A can be accommodated according to its features. If it cannot, then nothing happens; Agent A times out and the current cloning operation is aborted. If it can, then the receiving Agent System arranges to have the code of the clone of Agent A run by issuing an **Execute** request. This results in another agent being created, called Agent A’, which exists in a limited state.

3. Agent A’ determines if it has been cloned successfully and informs Agent A. If it has not, then Agent A terminates Agent A’ and the current cloning operation is aborted. An example of unsuccessful cloning could be where the resources requested by Agent A’ become unavailable during movement and subsequent execution.

4. If Agent A’ has been cloned successfully, then Agent A informs its Home Agent System of the location of new agent. This is necessary since it allows the owner of the mobile agent to keep track of the current location of all of its mobile agents.

5. The Home Agent System may accept or refuse this new agent location information according to features of Agent A (such as its credentials) or user restriction (such as the user not wanting another clone created). In such a case, Agent A terminates Agent A’ and the current cloning operation is aborted.

6. Now that all aspects of the cloning operation have been dealt with, Agent A issues a **Transfer** request to Agent A’. This will install in Agent A’ a similar identity and authority as Agent A possessed. Agent A’ now continues executing normally.
What is not shown in this protocol is Agent A' having to register itself with both its Home Agent System and also with the agent system upon which it has been cloned. These processes are explained in the Agent Management section of the FIPA 97 specification.

6.3. A Protocol for Agent Invocation

Invocation is the act of putting into effect an operation; in mobile agency, agent invocation refers to the creation of a new agent by an existing agent. Again, agent invocation is similar to both agent migration and agent cloning, except for the fact that there is no relationship between the invoking agent and the agent being invoked. It is simply a mechanism for creating and executing agents on potentially remote agent systems.

An example protocol for agent invocation is given in figure 6:

1. Agent A has decided to invoke Agent B on an Agent System, so it requests a language binding for Agent B from a Source (which could be an agent system or a locally available service).
2. If a language binding cannot be found or derived for Agent B, then the current invocation operation is aborted.
3. If it can, then Agent A issues a Move request to the Agent System. This request contains a suitable language binding and other information pertinent to the invocation, such as Agent B’s intended credentials and resource requirements.
4. The receiving Agent System processes the request and determines whether Agent B can be accommodated according to its features. If it cannot, then nothing happens; Agent A times out and the current invocation operation is aborted. If it can, then the Agent System arranges to have the code of Agent B run by issuing an Execute request. This results in another agent being created, called Agent B, which exists in a limited state.
5. Agent B determines if it has been invoked successfully and informs Agent A. If it has not, then Agent A terminates Agent B and the current invocation operation is aborted. An example of
unsuccessful invocation could be where the resources requested by Agent B become unavailable during movement and subsequent execution.

6. If Agent B has been invoked successfully, then Agent A informs its Home Agent System of the location of the new location. This is necessary since it allows the owner of the mobile agent to keep track of the current location of all of its mobile agents.

7. The Home Agent System may accept or refuse this new agent location information according to features of Agent A (such as its credentials) or user restriction (such as the user not wanting another agent invoked). In such a case, Agent A terminates Agent B and the current invocation operation is aborted.

8. Now that all aspects of the invocation operation have been dealt with, Agent A issues a Transfer request to Agent B. This will install in Agent B a subset of the authority that Agent A possessed but no identity, since Agent B has its own identity. Agent B now continues executing normally.

What is not shown in this protocol is Agent B having to register itself with both its Home Agent System and also with the agent system upon which it has been invoked. This process is explained the Agent Management section of the FIPA 97 specification.

The primary use of agent invocation is in the area of remote agent configuration; the key features being the ability to execute new agents on remote machines and the ability to control and influence their subsequent execution (see figure 7). In this diagram, there is one control agent which invokes agents on remote nodes and subsequently manages them. There is no requirement for invoked agents to be mobile once they have been created and executed, but if they do move, they have to inform their Home Agent System (which is generally the same agent system that the control agent executes upon). Typically, remote agent configuration is used to preconfigure a set of service providers across a network which represents interaction points for other stationary and mobile agents.

### 6.4. Support Issues

The following points identify how the issues of agent mobility can be handled by the specified primitives and additional agent system support:
Agent transfer is facilitated by the Move and Transfer primitives. When an agent decides to migrate or clone itself, it derives a suitable language binding appropriate to its intended form of migration and sends this to the agent system upon which the new agent is to execute. Agent transfer is a two stage operation; the beginning Move primitive to initiate migration, cloning or invocation and the final Transfer primitive to complete it. This allows the agent that began the transfer operation to monitor and control each stage of the process; the agent does not transfer identity or authority to the new agent until it is satisfied that the operation has been completed successfully. The underlying agent system needs to provide message transfer, code security and code execution support.

Message loss prevention is facilitated by the Transfer primitive. The underlying agent system needs to provide message buffering support for agents. When a mobile agent completes its migration to a new agent system with the Transfer primitive, it informs its Home Agent System and the agent system it is migrating from of its new agent system. This allows messages that were buffered for the agent either before it migrated or during its migration process to be forwarded its new location. However, this form of message forwarding can lead to a chain of message redirections being setup: a new link in the chain is created each time the mobile agent migrates. This is undesirable and potentially inefficient, especially if the chain grows to be long. To alleviate this, the agent system which has messages buffered for a mobile agent could check to see if the mobile agent was still executing on the agent system specified by the redirection. If so, then the messages can be forwarded as normal. If not, then the agent system could collapse the chain by following each redirection until it found the mobile agent, or it could contact the Home Agent System of the mobile agent. In any case, this process helps to ensure that the messages only have to be redirected once to reach the mobile agent.

With intermittently connected devices, further support is required from the agent system in the form of message buffering for agent systems that become disconnected. When a device is removed from the network, it needs an agent system that is on a permanently connected node to receive and buffer all messages on its behalf. In practice, this can be achieved by assigning a virtual agent system to each agent system that may be intermittently connected. Then, when an agent tries to send a message to an agent system that is disconnected, it fails and instead sends it to the well-known name of the corresponding virtual agent system\(^1\). When the device is reconnected, the agent system contacts its virtual agent system and downloads all of the pending messages.

Agent loss detection is facilitated by the Transfer primitive. When a mobile agent migrates, clones itself or invokes another agent, its Home Agent System is informed and has the chance to approve or decline the operation. In the case of approval, the Home Agent System is made aware of the new location of one of its existing mobile agents or is made aware of the existence of a new agent that it must additionally monitor. The Home Agent System can periodically check on the status of its agents since it has the addresses of the agent systems upon which they are executing. If an agent fails to respond or cannot be found, it may have been lost or terminated, in which case the Home Agent System can inquire with the appropriate agent system to find out what happened\(^2\).

To allow agents to be restarted in the event of their premature termination, checkpointing can be supported by each agent system. Basically, as each mobile agent moves to a new agent system, a copy of its language binding is written to the local storage area. This allows a Home Agent System to use agent invocation to restart at least the last incarnation of a mobile agent; if a Home Agent System keeps track of all of the agent systems that a mobile agent has ever visited during its lifetime, then it can potentially roll back to any previous incarnation.

---

\(^1\) The well-known name of a virtual agent system can be inferred from the name of the agent system that was originally contacted.

\(^2\) It may be the case that the mobile agent is in the process of completing a migration operation and has not yet had chance to inform its Home Agent System.
Agent management is facilitated by the Wait, Wake Up, Suspend, Resume, Destroy and Quit primitives. The management of agents can be achieved by allowing agent systems to control their execution in very specific or general ways. Specific ways involve directly affecting the execution of an agent by suspending it or destroying it. General ways involve monitoring and restricting access to system-level resources, such as processor cycles or memory usage. This support is crucial to controlling static and mobile agents, especially in environments where an agent has to pay for its resource usage. Therefore, system-level resource specification and control needs to be supported by the agent system.

7. Language Bindings

To promote the heterogeneity of mobile agents, agent systems must possess the ability to support a number of programming languages and mobile agent systems. However, how can the differences in functionality between various programming languages and various mobile agent systems be represented consistently across mobile agents and agent systems?

A solution advocated by this proposal concerns wrapping the specific features of a programming language or mobile agent system into an abstraction, called a language binding. A language binding allows a mobile agent written in a particular programming language or for a particular mobile agent system to express certain features and requirements. The following attributes could be used to represent these:

- **System** specifies which programming language or mobile agent system is required in order to execute the mobile agent, for example, Java, Tacoma, Project Mole, etc.
- **Version** specifies which version of the programming language or mobile agent system is required in order to execute the mobile agent.
- **Format** specifies the type of agent code which is being transferred, for example, byte code, object code, source code, etc.
- **Form** specifies to nature of the agent state being transferred, for example, closure, serialised object, thread continuation, etc.
- **Code** is a data block representing the agent state (including code and possibly data and execution state) being transferred.
- **Data** is a data block representing the agent data being transferred.

Other keywords may be required in order to fully support all of the programming language and mobile agent system possibilities and configurations. Figure 8 gives an example language binding for a mobile agent. This particular agent is written in APRIL (Agent Process Interaction Language [9]) version 4 and is represented as a byte code closure which already contains the data of the agent. Programming languages which do not support closures may require the code of an agent to be stored separately from its data, hence the need for both a code and a data field.

```
(:language-binding
  (:system "APRIL")
  (:version "4.00")
  (:format byte-code)
  (:form closure)
  (:code "...")
  (:data ")
```

**Figure 8 : Example Language Binding for an APRIL Mobile Agent**
The value of each attribute within the language binding can be used by agent systems to match the requirements of a mobile agent against their supported languages and systems. Therefore, multiple language bindings will be available within an agent system corresponding to the types and forms of mobile agents that it can support\(^1\). In some cases, the form of a mobile agent may require the agent system to assist it through to actual execution. For example, for source code, the agent system may have to compile the code before it can invoke an executor to run it.

In this way, heterogeneity of mobile agent forms can be supported across agent systems. A mobile agent also has a mechanism by which it can be assured of being able to execute on a remote agent system before it decides to move. Indeed, mobile agents may negotiate with many remote agents systems by sending them full or partial language binding specifications to determine to which systems they can move.

8. Critique

This document has put forward a proposal for incorporating mobility into the FIPA 98 specification. The following subsections describe how it meets the proposed specification for Agent Management Support for Mobility in the Fourth Call for Proposals [5].

8.1. Normative Topics

The proposal addresses these normative points:

- The minimum set of required primitives to support agent mobility have been defined in sub-section 5.2: agent identity management (Transfer), creation (Move, Execute) and destruction (Quit). It has been shown that these primitives are enough to describe both the three forms of basic agent mobility identified in section 3 (agent migration, agent cloning and agent invocation) and also the agent mobility protocols given in section 6.

- The monitoring and management of mobile agents has been outlined in sub-sections 4.2 and 6.4, including the need for management primitives to control agents (based upon the agent life cycle given in Part 1 of the FIPA 97 specification). It has been shown that a mobile agent can be controlled explicitly through the use of the primitives identified in sub-section 4.1 (figure 1) and implicitly through resource monitoring and control. Support is required at the agent system level to support resource specification and management.

- This proposal advocates the need for agent mobility to support intermittently connected devices, such as portable computers and PDAs and has shown how they can be supported with assistance from the agent system. In particular, support is required to allow asynchronous message passing, message buffering and message redirection, such as those proposed in [10].

- Agent system and mobile agent requirements have been addressed in section 5 and also in section 7. Language bindings can be used by both mobile agents and also agent systems to reason about the requirements that mobile agents have and the facilities that agent systems offer. Additionally, language bindings can be used by agent systems to implement checkpointing and roll back for mobile agents in the event that a future version of the agent terminates prematurely.

- Mobile agent requirements for code mobility, closure mobility and process mobility have been addresses in sub-section 5.1 and section 7. The relationship to the actual form of migration that can be allowed is closely tied to the facilities offered by individual programming languages and mobile agent systems. Language bindings allow these implementation-oriented forms of mobility to be expressed in a flexible manner that is not prescriptive.

- The mobility protocols required to support the three forms of mobility identified in section 3 have been defined in section 6. These protocols are based upon the assumption that the mobile agent is an active party in the migration process. In this way, mobility consists of two basic actions;

\(^1\) This will probably be dictated by which programming languages and mobile agent systems are installed on the agent system.
movement of a mobile agent’s state (Move) and movement of a mobile agent’s identity and authority (Transfer). This is advantageous since it allows the mobile agent to direct the course of migration in unexpected circumstances. However, this is powerful enough to allow various other types of protocols to be defined, including those in which the mobile agent is a passive entity in the mobility operation, as exemplified by the Telescript ‘go’ operation.

- Agent security has been highlighted throughout this proposal and is seen as a basic requirement of the agent system, because security is not just an issue for mobile agents but for agents in general. Mechanisms need to be in place to allow agent management, code verification, user authentication, digital signing, encryption, application-level authorisation and access rights, and system-level resource management.

8.2. Informative Topics

The proposal addresses these informative points:

- Application domains in which mobile agents are being developed were given in sections 6.1 and 6.3. Mobility is being used in such application areas as information retrieval and management, electronic commerce, remote agent configuration and in providing support for mobile computers.

- The distinction between mobile agents and mobile objects has not been addressed directly by this proposal. The main reasons behind this are twofold. Firstly, since mobile agents are agents plus mobility and mobile objects are objects plus mobility, the similarities between mobile agents and mobile objects becomes too great to discuss. Secondly, mobile objects are considered to be an implementation technology for mobile agents.

- The main differences between this proposal and the OMG’s MAF proposal [8] lies in the flexibility of the forms of mobility and mobility protocols provided. In MAF, the form of mobility is limited to agent transfer (agent migration) and remote agent creation (remote agent invocation). Also, the only mobility protocol supported is agent transfer, which is reduced to a single operation where the mobile agent is a passive entity. The advantage of this proposal is that it is rich enough to support many forms of mobility and mobility protocols, including the one advocated by MAF. Additionally, MAF is based around CORBA which only supports a synchronous client-server communication model; agents are more suited to an asynchronous peer-to-peer communication model to maximise the number of potential interactions that they can have.

9. References


1. As defined by the ECOOP Workshop on Mobile Object Systems.
2. There are mechanisms available in CORBA 2.0 to achieve a style of asynchronous communication. However, an asynchronous messaging service, called Message Oriented Middleware (MOM), is being proposed for CORBA version 3.0.


## Appendix A

### A.1. move

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported By</td>
<td>AMS</td>
</tr>
<tr>
<td>Description</td>
<td>An agent issues a move request to transfer all or part of itself to a remote AMS. There is an intended future commitment made by the moving agent that it will transfer some identity and authority if the remote AMS approves the move. However, the AMS may refuse to accept the move request due to lack of language-binding support or other local restrictions. When an agent applies to move to a remote AMS, an agent description must be supplied containing values for all of the mandatory attributes of the agent description.</td>
</tr>
<tr>
<td>Content</td>
<td>fipa-man-ams-agent-description</td>
</tr>
<tr>
<td>FIPA Protocol</td>
<td>fipa-request</td>
</tr>
<tr>
<td>Example</td>
<td>(request&lt;br&gt;  :sender an-agent@async://fipa.org/acc&lt;br&gt;  :receiver an-ams@async://fipa.org/acc&lt;br&gt;  :content&lt;br&gt;     (action an-ams@async://fipa.org/acc&lt;br&gt;         (move&lt;br&gt;             (:agent-name an-agent@async://fipa.org/acc)&lt;br&gt;             (:address async://fipa.org/acc)&lt;br&gt;             (:agent-requirements ...)&lt;br&gt;             (:language-binding ...)&lt;br&gt;             (:ownership fipa.org))&lt;br&gt;     :protocol fipa-request&lt;br&gt;     :ontology fipa-agent-management)</td>
</tr>
<tr>
<td>Failure Reasons</td>
<td>language-binding-unsupported</td>
</tr>
<tr>
<td></td>
<td>requirements-not-available</td>
</tr>
<tr>
<td></td>
<td>agent-already-present</td>
</tr>
</tbody>
</table>
A.2. transfer

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported By</td>
<td>AMS</td>
</tr>
<tr>
<td>Description</td>
<td>An agent issues a transfer request to move part or all of its identity and authority to another agent on a remote AMS. However, the AMS may refuse to accept the transfer request for security reasons. When an agent applies to transfer to a remote AMS an agent description must be supplied containing values for all of the mandatory attributes of the agent description.</td>
</tr>
<tr>
<td>Content</td>
<td>fipa-man-ams-agent-description</td>
</tr>
<tr>
<td>Protocol</td>
<td>fipa-request</td>
</tr>
<tr>
<td>Example</td>
<td>(request :sender an-agent@async://fipa.org/acc :receiver an-ams@async://fipa.org/acc :content (action an-ams@async://fipa.org/acc (transfer (:agent-name a-new-agent@async://fipa.org/acc (:signature &quot;...&quot;) (:ownership fipa.org)) :protocol fipa-request :ontology fipa-agent-management)</td>
</tr>
<tr>
<td>Failure Reasons</td>
<td>agent-not-present This failure occurs if the recipient agent specified by the :agent-name parameter is not present within the remote AMS.</td>
</tr>
<tr>
<td></td>
<td>not-agent-owner This failure occurs if the agent requesting the transfer does not own the recipient agent specified by the :agent-name parameter.</td>
</tr>
<tr>
<td></td>
<td>signature-not-valid This failure occurs if the signature is not valid.</td>
</tr>
</tbody>
</table>
A.3. fipa-man-ams-agent-description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:agent-name</td>
<td>Denotes the GUID of the agent.</td>
</tr>
<tr>
<td>:address</td>
<td>Denotes the communication address of the agent.</td>
</tr>
<tr>
<td>:signature</td>
<td>Denotes the secure and encrypted signature of the agent.</td>
</tr>
<tr>
<td>:delegate-agent</td>
<td>Denotes the agent that has been delegated as a recipient of all messages intended for the agent specified in the :agent-name parameter.</td>
</tr>
<tr>
<td>:forward-address</td>
<td>Denotes the agent to which all future messages should be sent for the agent specification in the :agent-name parameter.</td>
</tr>
<tr>
<td>:agent-requirements</td>
<td>Denotes the requirements of the agent. See fipa-man-requirements-description.</td>
</tr>
<tr>
<td>:language-binding</td>
<td>Denotes the language-binding of the agent. See fipa-man-language-binding-description.</td>
</tr>
<tr>
<td>:ownership</td>
<td>Denotes the owner who is responsible for the agent.</td>
</tr>
<tr>
<td>:ap-state</td>
<td>Denotes the life-cycle state of the agent.</td>
</tr>
</tbody>
</table>

A.4. fipa-man-requirements-description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:requirement-type</td>
<td>Denotes the unique requirement type.</td>
</tr>
<tr>
<td>:requirement-ontology</td>
<td>Denotes the ontology of the requirement.</td>
</tr>
<tr>
<td>:requirement-description</td>
<td>Denotes the description of the requirement. This could be a complex structure that is defined in terms of the ontology given in the :requirement-ontology parameter.</td>
</tr>
<tr>
<td>:requirement-condition</td>
<td>Denotes the conditions in which the requirement can be fulfilled.</td>
</tr>
</tbody>
</table>
A.5. fipa-man-language-binding-description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:system</td>
<td>Denotes the programming language or mobile agent system required by the language-binding.</td>
</tr>
<tr>
<td>:version</td>
<td>Denotes the version of the programming language or mobile agent system required by the language-binding.</td>
</tr>
<tr>
<td>:format</td>
<td>Denotes what format the :code parameter in the language-binding takes, for example, byte-code, source-code, object-code, etc.</td>
</tr>
<tr>
<td>:form</td>
<td>Denotes what can be found in the :code parameter in the language-binding, for example, closure, serialised-object, thread-continuation, etc.</td>
</tr>
<tr>
<td>:code</td>
<td>Denotes the agent state of the agent.</td>
</tr>
<tr>
<td>:data</td>
<td>Denotes any data associated with the agent.</td>
</tr>
</tbody>
</table>

A.6. fipa-man-ams-agent-description Mandatory/Optional Parameter List

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
<td>transfer</td>
</tr>
<tr>
<td>:agent-name</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:address</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:signature</td>
<td>Optional</td>
</tr>
<tr>
<td>:delegate-agent</td>
<td>Optional</td>
</tr>
<tr>
<td>:forward-address</td>
<td>Optional</td>
</tr>
<tr>
<td>:agent-requirements</td>
<td>Optional</td>
</tr>
<tr>
<td>:language-binding</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:ownership</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:ap-state</td>
<td>Optional</td>
</tr>
</tbody>
</table>
A.7. fipa-man-requirement-description Mandatory/Optional Parameter List

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
<td></td>
</tr>
<tr>
<td>:requirement-type</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:requirement-ontology</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:requirement-description</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:requirement-condition</td>
<td>Optional</td>
</tr>
</tbody>
</table>

A.8. fipa-man-language-binding-description Mandatory/Optional Parameter List

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>move</td>
<td></td>
</tr>
<tr>
<td>:system</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:version</td>
<td>Optional</td>
</tr>
<tr>
<td>:format</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:form</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:code</td>
<td>Mandatory</td>
</tr>
<tr>
<td>:data</td>
<td>Optional</td>
</tr>
</tbody>
</table>