FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS

FIPA Agent Management Specification

Document title	FIPA Agent Management Specification		
Document number	XC00023H	**Document source**	FIPA Agent Management
Document status	Experimental	**Date of this status**	2001/10/03
Supersedes	FIPA00002, FIPA00017, FIPA00019		
Contact	fab@fipa.org		
Change history			
2000/07/31	Approved for Experimental; removed conflicting description of :`name` parameter in the `agent-identifier` object		
2000/08/28	Made all parameters of the DF, AMS and Service descriptions optional to allow them to be used with the `search` function		
2001/08/10	Line numbering added		

Geneva, Switzerland

Notice

Use of the technologies described in this specification may infringe patents, copyrights or other intellectual property rights of FIPA Members and non-members. Nothing in this specification should be construed as granting permission to use any of the technologies described. Anyone planning to make use of technology covered by the intellectual property rights of others should first obtain permission from the holder(s) of the rights. FIPA strongly encourages anyone implementing any part of this specification to determine first whether part(s) sought to be implemented are covered by the intellectual property of others, and, if so, to obtain appropriate licenses or other permission from the holder(s) of such intellectual property prior to implementation. This specification is subject to change without notice. Neither FIPA nor any of its Members accept any responsibility whatsoever for damages or liability, direct or consequential, which may result from the use of this specification.

Foreword

The Foundation for Intelligent Physical Agents (FIPA) is an international organization that is dedicated to promoting the industry of intelligent agents by openly developing specifications supporting interoperability among agents and agent-based applications. This occurs through open collaboration among its member organizations, which are companies and universities that are active in the field of agents. FIPA makes the results of its activities available to all interested parties and intends to contribute its results to the appropriate formal standards bodies.

The members of FIPA are individually and collectively committed to open competition in the development of agent-based applications, services and equipment. Membership in FIPA is open to any corporation and individual firm, partnership, governmental body or international organization without restriction. In particular, members are not bound to implement or use specific agent-based standards, recommendations and FIPA specifications by virtue of their participation in FIPA.

The FIPA specifications are developed through direct involvement of the FIPA membership. The status of a specification can be either Preliminary, Experimental, Standard, Deprecated or Obsolete. More detail about the process of specification may be found in the FIPA Procedures for Technical Work. A complete overview of the FIPA specifications and their current status may be found in the FIPA List of Specifications. A list of terms and abbreviations used in the FIPA specifications may be found in the FIPA Glossary.

FIPA is a non-profit association registered in Geneva, Switzerland. As of January 2000, the 56 members of FIPA represented 17 countries worldwide. Further information about FIPA as an organization, membership information, FIPA specifications and upcoming meetings may be found at http://www.fipa.org/.

Contents

1 Scope

This document is part of the FIPA specifications covering agent management for inter-operable agents. This specification incorporates and further enhances the FIPA 98 Agent Management Specification [FIPA00002]. The FIPA Agent Message Transport Specification [FIPA00067] represent a companion specification.

This document contains specifications for agent management including agent management services, agent management ontology and agent platform message transport. This document is primarily concerned with defining open standard interfaces for accessing agent management services. The internal design and implementation of intelligent agents and agent management infrastructure is not mandated by FIPA and is outside the scope of this specification.

The document provides a series of examples to illustrate the agent management functions defined.

2 Agent Management Reference Model

Agent management provides the normative framework within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents.

The entities contained in the reference model (see *Figure 1*) are logical capability sets (that is, services) and do not imply any physical configuration. Additionally, the implementation details of individual APs and agents are the design choices of the individual agent system developers.

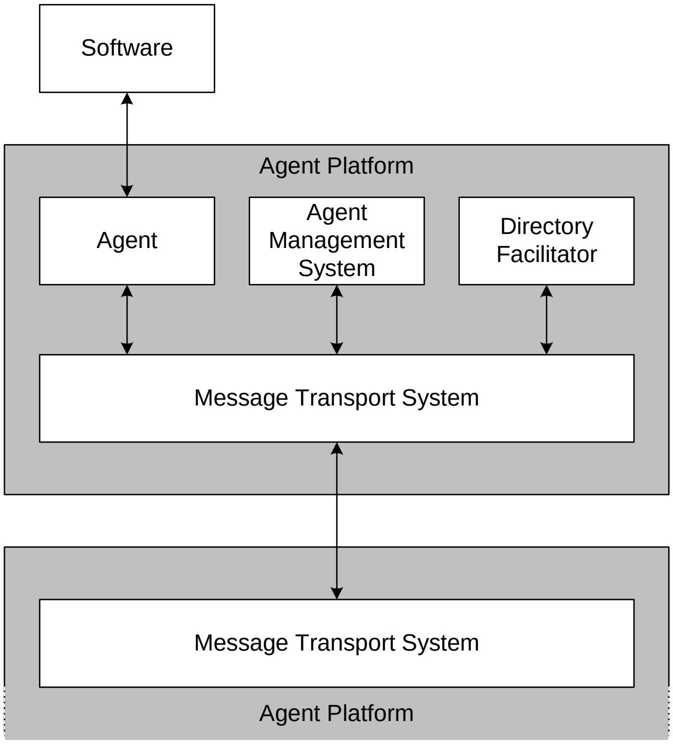

Figure 1: Agent Management Reference Model

The agent management reference model consists of the following logical components, each representing a capability set (these can be combined in physical implementations of APs):

- An **Agent** is the fundamental actor on an AP which combines one or more service capabilities into a unified and integrated execution model that may include access to external software, human users and communications facilities. An agent may have certain resource brokering capabilities for accessing software (see [FIPA00079]).

  An agent must have at least one owner, for example, based on organisational affiliation or human user ownership, and an agent may support several notions of identity. An Agent Identifier (AID) labels an agent so that it may be distinguished unambiguously within the Agent Universe. An agent may be registered at a number of transport addresses at which it can be contacted and it may have certain resource brokering capabilities for accessing software.

- A **Directory Facilitator (DF)** is a mandatory component of the AP. The DF provides yellow pages services to other agents. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Multiple DFs may exist within an AP and may be federated.

- An **Agent Management System (AMS)** is a mandatory component of the AP. The AMS exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS maintains a directory of AIDs which contain transport addresses (amongst other things) for agents registered with the AP. The AMS offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID.

- An **Message Transport Service (MTS)** is the default communication method between agents on different APs (see [FIPA00067]).

- An **Agent Platform (AP)** provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system, agent support software, FIPA agent management components (DF, AMS and MTS) and agents.

  The internal design of an AP is an issue for agent system developers and is not a subject of standardisation within FIPA. AP's and the agents which are native to those APs, either by creation directly within or migration to the AP, may use any proprietary method of inter-communication.

  It should be noted that the concept of an AP does not mean that all agents resident on an AP have to be co-located on the same host computer. FIPA envisages a variety of different APs from single processes containing lightweight agent threads, to fully distributed APs built around proprietary or open middleware standards.

  FIPA is concerned only with how communication is carried out between agents who are native to the AP and agents outside the AP or agents who dynamically register with an AP. Agents are free to exchange messages directly by any means that they can support.

- **Software** describes all non-agent, executable collections of instructions accessible through an agent. Agents may access software, for example, to add new services, acquire new communications protocols, acquire new security protocols/algorithms, acquire new negotiation protocols, access tools which support migration, etc.

3 Agent Naming

The FIPA agent naming reference model identifies an agent through an extensible collection of parameter-value pairs, called an Agent Identifier (AID). An AID comprises[1]:

- A name.

- Other parameters, such as transport addresses, name resolution service addresses, and so on.

The extensible nature of an AID allows it to be augmented to accommodate other requirements, such as social names, nick names, roles, etc. which can then be attached to services within the AP.

AIDs are primarily intended to be used to identify agents inside the envelope of a message, specifically within the `:to` and `:from` parameters (see [FIPA00067]). The definition of the AID object and its parameters is given in section *6.1.1, Agent Identifier Description*.

The parameter values of an AID can be edited or modified by an agent, for example, to update the sequence of name resolution servers or transport addresses in an AID. However, the mandatory parameters can only be changed by the agent to whom the AID belongs.

The `:name` parameter of an AID is a globally unique identifier that can be used as a unique referring expression of the agent. One of the simplest mechanisms is to construct it from the actual name of the agent and its home agent platform address[2] (HAP), separated by the '@' character.

3.1 Transport Addresses

A transport address is a physical address at which an agent can be contacted and is usually specific to a Message Transport Protocol. A given agent may support many methods of communication and can put multiple transport address values in the `:addresses` parameter of an AID.

The EBNF syntax of a transport addresses is the same as for a URL given in [RFC2396]. [FIPA00067] describes the semantics of message delivery with regard to transport addresses.

3.2 Name Resolution

Name resolution is a service that is provided by the AMS through the `search` function. The `:resolvers` parameter of the AID contains a sequence of AIDs at which the AID of the agent can ultimately be resolved into a transport address or set of transport address.

An example name resolution pattern might be:

1. AgentA wishes to send a message to AgentB, whose AID is:

```
(agent-identifier
  :name AgentB@bar.com
  :resolvers (sequence
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc))))
```

   and AgentA wishes to know additional transport addresses that have been given for AgentB.

[1] The name of an agent is immutable and cannot be changed during the lifetime of the agent; the other parameters in the AID of an agent can be changed.

[2] The HAP of an agent is the AP on which the agent was created.

2. Therefore, AgentA can send a `search` request to the first agent specified in the `:resolvers` parameter which is typically an AMS. In this example, the AMS at `foo.com`.

3. If the AMS at `foo.com` has AgentB registered with it, then it returns a `result` message containing the AMS agent description of AgentB; if not, then a `failed` message is returned.

4. Upon receipt of the `result` message, AgentA can extract the `agent-identifier` parameter of the `ams-agent-description` and then extract the `:addresses` parameter of this to determine the transport address(es) of AgentB.

5. AgentA can now send a message to AgentB by inserting the `:addresses` parameter into the AID of AgentB.

4 Agent Management Services

4.1 Directory Facilitator

4.1.1 Overview

A DF is a mandatory component of an AP that provides a yellow pages directory service to agents. It is the trusted, benign custodian of the agent directory. It is trusted in the sense that it must strive to maintain an accurate, complete and timely list of agents. It is benign in the sense that it must provide the most current information about agents in its directory on a non-discriminatory basis to all authorised agents. At least one DF must be resident on each AP (the default DF). However, an AP may support any number of DFs and DFs may register with each other to form federations.

Every agent that wishes to publicise its services to other agents, should find an appropriate DF and request the **registration** of its agent description. There is no intended future commitment or obligation on the part of the registering agent implied in the act of registering. For example, an agent can refuse a request for a service which is advertised through a DF. Additionally, the DF cannot guarantee the validity or accuracy of the information that has been registered with it, neither can it control the life cycle of any agent. An object description must be supplied containing values for all of the mandatory parameters of the description. It may also supply optional and private parameters, containing non-FIPA standardised information that an agent developer might want included in the directory. The **deregistration** function has the consequence that there is no longer a commitment on behalf of the DF to broker information relating to that agent. At any time, and for any reason, the agent may request the DF to **modify** its agent description.

An agent may **search** in order to request information from a DF. The DF does not guarantee the validity of the information provided in response to a search request, since the DF does not place any restrictions on the information that can be registered with it. However, the DF may restrict access to information in its directory and will verify all access permissions for agents which attempt to inform it of agent state changes.

The default DF on an AP has a reserved AID of:

```
(agent-identifier
  :name df@hap
  :addresses (sequence hap_transport_address))
```

4.1.2 Management Functions Supported by the Directory Facilitator

In order to access the directory of agent descriptions managed by the DF, each DF must be able to perform the following functions, when defined on the domain of objects of type `df-agent-description` in compliance with the semantics described in section *6.1.2, Directory Facilitator Agent Description*:

- `register`
- `deregister`
- `modify`
- `search`

4.1.3 Federated Directory Facilitators

The DF encompasses a search mechanism that searches first locally and then extends the search to other DFs, if allowed. The default search mechanism is assumed to be a depth-first search across DFs. For specific purposes, optional constraints can be used as described in section *6.1.4, Search Constraints* such as the number of answers (`:df-search-results`). The federation of DFs for extending searches can be achieved by DFs registering with each other with `fipa-df` as the value of the `:type` parameter in the `service-description`.

4.2 Agent Management System

4.2.1 Overview

An AMS is a mandatory component of the AP and only one AMS will exist in a single AP. The AMS is responsible for managing the operation of an AP, such as the creation of agents, the deletion of agents, deciding whether an agent can dynamically register with the AP and overseeing the migration of agents to and from the AP (if agent mobility is supported by the AP). Since different APs have different capabilities, the AMS can be queried to obtain a description of its AP. A life cycle is associated with each agent on the AP (see section *5.1, Agent Life Cycle*) which is maintained by the AMS.

The AMS represents the managing authority of an AP and if the AP spans multiple machines, then the AMS represents the authority across all machines. An AMS can request that an agent performs a specific management function, such as `quit` (that is, terminate all execution on its AP) and has the authority to forcibly enforce the function if such a request is ignored.

The AMS maintains an index of all the agents that are currently resident on an AP, which includes the AID of agents. Residency of an agent on the AP implies that the agent has been registered with the AMS. Each agent, in order to comply with the FIPA reference model, must **register** with the AMS of its HAP. Registration with the AMS, implies authorisation to access the MTS of the AP in order to send or receive messages. The AMS will check the validity of the passed agent description and, in particular, the local uniqueness of the agent name in the AID.

Agent descriptions can be later **modified** at any time and for any reason. Modification is restricted by authorisation of the AMS. The life of an agent with an AP terminates with its **deregistration** from the AMS. After deregistration, the AID of that agent can be removed by the directory and can be made available to other agents who should request it.

Agent description can be **searched** with the AMS and access to the directory of `ams-agent-descriptions` is further controlled by the AMS; no default policy is specified by this specification.

The AMS is also the custodian of the AP description that can be retrieved by requesting the action `get-description.`

The AMS on an AP has a reserved AID of:

```
(agent-identifier
  :name ams@hap
  :addresses (sequence hap_transport_address))
```

4.2.2 Management Functions Supported by the Agent Management System

An AMS must be able to perform the following functions, in compliance with the semantics described in section *6.1.5, Agent Management System Agent Description* (the first four functions are defined within the scope of the AMS, only on the domain of objects of type `ams-agent-description` and the last on the domain of objects of type `ap-description`):

- `register`

- `deregister`

- `modify`

- `search`

- `get-description`

In addition to the management functions exchanged between the AMS and agents on the AP, the AMS can instruct the underlying AP to perform the following operations:

- Suspend agent,

- Terminate agent,

- Create agent,

- Resume agent execution,

- Invoke agent,

- Execute agent, and,

- Resource management.

4.2.3 Management Functions Supported by Agents

Mandatory agent functions:

- `quit`

This function is described in section *6.2.6, Terminate an Agent*.

4.3 Message Transport Service

The Message Transport Service (MTS) delivers messages between agents within an AP and to agents resident on other APs. All FIPA agents have access to at least one MTS and only messages addressed to an agent can be sent to the MTS. See [FIPA00067] for more information on the MTS.

5 Agent Platform

5.1 Agent Life Cycle

FIPA agents exist physically on an AP and utilise the facilities offered by the AP for realising their functionalities. In this context, an agent, as a physical software process, has a physical life cycle that has to be managed by the AP.

The life cycle of a FIPA agent is (see *Figure 2*):

- **AP Bounded**
  An agent is physically managed within an AP and the life cycle of a static agent is therefore always bounded to a specific AP.

- **Application Independent**
  The life cycle model is independent from any application system and it defines only the states and the transitions of the agent service in its life cycle.

- **Instance-Oriented**
  The agent described in the life cycle model is assumed to be an instance (that is, an agent which has unique name and is executed independently).

- **Unique**
  Each agent has only one AP life cycle state at any time and within only one AP.

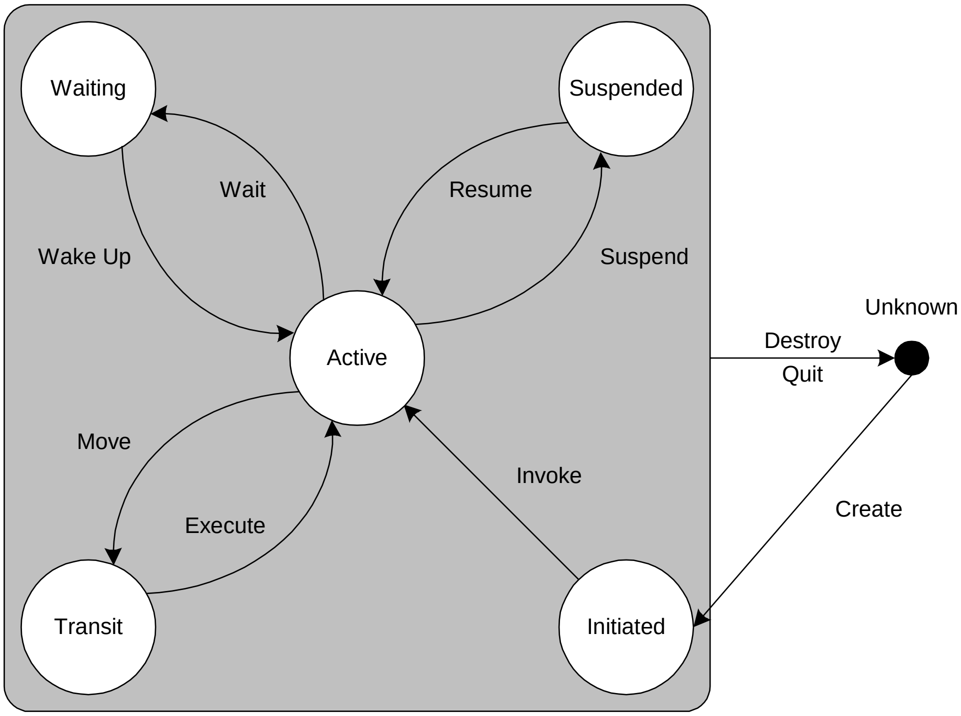

Figure 2: Agent Life Cycle

The followings are the responsibility that an AMS, on behalf of the AP, has with regard to message delivery in each state of the life cycle of an agent:

- **Active**
  The MTS delivers messages to the agent as normal.

- **Initiated/Waiting/Suspended**
  The MTS either buffers messages until the agent returns to the active state or forwards messages to a new location (if a forward is set for the agent).

- **Transit**
  The MTS either buffers messages until the agent becomes active (that is, the move function failed on the original AP or the agent was successfully started on the destination AP) or forwards messages to a new location (if a forward is set for the agent). Notice that Only mobile agents can enter the **Transit** state. This ensures that a stationary agent executes all of its instructions on the node where it was invoked.

- **Unknown**
  The MTS either buffers messages or rejects them, depending upon the policy of the MTS and the transport requirements of the message.

The state transitions of agents can be described as:

- **Create**
  The creation or 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 AMS 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 AMS.

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

- **Wait**
  Puts an agent in a waiting state. This can only be initiated by an agent.

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

The following two transitions are only used by mobile agents (see [FIPA00005]):

- **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 AMS.

5.2 Agent Registration

There are three ways in which an agent can be registered with an AMS:

- The agent was created on the AP.

- The agent migrated to the AP, for those APs which support agent mobility (see [FIPA00005]).

- The agent explicitly registered with the AP, assuming that the AP both supports dynamic registration and is willing to register the new agent. Dynamic registration is where an agent which has a HAP wishes to register on another AP as a local agent.

Agent registration involves registering an AID with the AMS. When an agent is either created or dynamically registers with an AP, the agent is registered with the AMS, for example by using the `register` function. In the following example, an agent called *discovery-agent* is registering dynamically with an AP located at `foo.com`. The agent *discovery-agent* was created on the AP (that is, *discovery-agent*'s HAP) at `bar.com` and requests that the AMS registers it.

For example:

```
(request
  :sender
    (agent-identifier
      :name discovery-agent@bar.com
      :addresses (sequence iiop://bar.com/acc))
  :receiver (set
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :ontology FIPA-Agent-Management
  :language FIPA-SL0
  :protocol FIPA-Request
  :content
    (action
      (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (:ams-description
          :name
            (agent-identifier
              :name discovery-agent@bar.com
              :addresses (sequence iiop://bar.com/acc))
           ...)))
```

It should be noted that the `:addresses` parameter of the AID represents the transport address(es) that the agent would like any messages directed to (see [FIPA00067] for information on how the MTS deals with this). In the above example, the agent *discovery-agent* registers itself with the `foo.com` AP but by virtue of specifying a different transport address in the `:addresses` parameter of its AID, messages that arrive at `foo.com` will be forwarded to `bar.com`.

6 Agent Management Ontology

6.1 Object Descriptions

This section describes a set of frames, that represent the classes of objects in the domain of discourse within the framework of the `FIPA-Agent-Management` ontology.

The following terms are used to describe the objects of the domain:

- **Frame**. This is the mandatory name of this entity, that must be used to represent each instance of this class.

- **Ontology**. This is the name of the ontology, whose domain of discourse includes the parameters described in the table.

- **Parameter**. This is the mandatory name of a parameter of this frame.

- **Description**. This is a natural language description of the semantics of each parameter.

- **Presence**. This indicates whether each parameter is mandatory or optional.

- **Type**. This is the type of the values of the parameter: Integer, Word, String, URL, Term, Set or Sequence.

- **Reserved Values**. This is a list of FIPA-defined constants that can assume values for this parameter.

6.1.1 Agent Identifier Description

This type of object represents the identification of the agent.

Frame	`agent-identifier`
Ontology	`FIPA-Agent-Management`

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**

`name`	The symbolic name of the agent.	Mandatory	`Word`	`df@`*`hap`*
				`ams@`*`hap`*

`addresses`	A sequence of ordered transport addresses where the agent can be contacted. The order implies a preference relation of the agent to receive messages over that address.	Optional	Sequence of `URL`	
`resolvers`	A sequence of ordered AIDs where name resolution services for the agent can be contacted. The order in the sequence implies a preference in the list of resolvers.	Optional	Sequence of `agent-identifier`	

6.1.2 Directory Facilitator Agent Description

This type of object represents the description that can be registered with the DF yellow-page service.

Frame	df-agent-description
Ontology	FIPA-Agent-Management

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
name	The identifier of the agent.	Optional	agent-identifier	
services	A list of services supported by this agent.	Optional	Set of service-description	
protocol	A list of interaction protocols supported by the agent.	Optional	Set of String	See [FIPA00025]
ontology	A list of ontologies known by the agent.	Optional	Set of String	FIPA-Agent-Management
language	A list of content languages known by the agent.	Optional	Set of String	FIPA-SL FIPA-SL0 FIPA-SL1 FIPA-SL2

6.1.3 Service Description

This type of object represents the description of each service registered with the DF.

Frame	service-description
Ontology	FIPA-Agent-Management

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
name	The name of the service.	Optional	String	
type	The type of the service.	Optional	String	fipa-df fipa-ams
protocol	A list of interaction protocols supported by the service.	Optional	Set of String	
ontology	A list of ontologies supported by the service.	Optional	Set of String	FIPA-Agent-Management
language	A list of content languages supported by the service.	Optional	Set of String	
ownership	The owner of the service	Optional	String	
properties	A list of properties that discriminate the service.	Optional	Set of property	

6.1.4 Search Constraints

This type of object represents a set of constraints to limit the function of searching within a directory.

Frame	`search-constraints`
Ontology	`FIPA-Agent-Management`

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
`max-depth`	The maximum depth of propagation of the search to federated directories. This value should not be negative.	Optional	`Integer`	
`max-results`	The maximum number of results to return for the search. This value should not be negative.	Optional	`Integer`	

6.1.5 Agent Management System Agent Description

This type of object represents the agent descriptions treated by an AMS agent.

Frame	`ams-agent-description`
Ontology	`FIPA-Agent-Management`

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
`name`	The identifier of the agent.	Optional	`agent-identifier`	
`ownership`	The owner of the agent.	Optional	`String`	

`state`	The life cycle state of the agent.	Optional	`String`	`initiated`
				`active`
				`suspended`
				`waiting`
				`transit`

6.1.6 Agent Platform Description

Frame	`ap-description`
Ontology	`FIPA-Agent-Management`

Parameter	**Description**	**Presence**	**Type**	**Reserved Values**
`name`	The name of the AP.	Mandatory	`String`	
`dynamic`	The support for dynamic registration of the AP.	Optional	`Boolean`	
`mobility`	The support for mobility of the AP.	Optional	`Boolean`	
`transport-profile`	The description MTS capabilities of the AP.	Optional	`ap-transport-description`	See [FIPA00067]

6.1.7 Property Template

This is a special object that is useful for specifying parameter/value pairs.

Frame **Ontology**	`property` `FIPA-Agent-Management`

Parameter	Description	Presence	Type	Reserved Values
`name`	The name of the property.	Mandatory	`String`	
`value`	The value of the property	Mandatory	`Term`	

6.2 Function Descriptions

The following tables define usage and semantics of the functions that are part of the `FIPA-Agent-Management` ontology and that are supported by the agent management services and agents on the AP.

The following terms are used to describe the functions of the `FIPA-Agent-Management` domain:

- **Function**. This is the symbol that identifies the function in the ontology.

- **Ontology**. This is the name of the ontology, whose domain of discourse includes the function described in the table.

- **Supported by**. This is the type of agent that supports this function.

- **Description**. This is a natural language description of the semantics of the function.

- **Domain**. This indicates the domain over which the function is defined. The arguments passed to the function must belong to the set identified by the domain.

- **Range**. This indicates the range to which the function maps the symbols of the domain. The result of the function is a symbol belonging to the set identified by the range.

- **Arity**. This indicates the number of arguments that a function takes. If a function can take an arbitrary number of arguments, then its arity is undefined.

6.2.1 Registration of an Object with an Agent

Function	`register`
Ontology	`FIPA-Agent-Management`
Supported by	DF and AMS
Description	The execution of this function has the effect of registering a new object into the knowledge base of the executing agent. The DF or AMS description supplied must include a valid AID.
Domain	`df-agent-description` / `ams-agent-description`
Range	The execution of this function results in a change of the state, but it has no explicit result. Therefore there is no range set.
Arity	1

6.2.2 Deregistration of an Object with an Agent

Function	`deregister`
Ontology	`FIPA-Agent-Management`
Supported by	DF and AMS
Description	An agent may deregister an object in order to remove all of its parameters from a directory. The DF or AMS description supplied must include a valid AID.
Domain	`df-agent-description` / `ams-agent-description`
Range	The execution of this function results in a change of the state, but it has no explicit result. Therefore there is no range set.
Arity	1

6.2.3 Modification of an Object Registration with an Agent

Function	`modify`
Ontology	`FIPA-Agent-Management`
Supported by	DF and AMS
Description	An agent may make a modification in order to change its object registration with another agent. The argument of a `modify` function will replace the existing object description stored within the executing agent. The DF or AMS description supplied must include a valid AID.
Domain	`df-agent-description` / `ams-agent-description`
Range	The execution of this function results in a change of the state, but it has no explicit result. Therefore there is no range set.
Arity	1

6.2.4 Search for an Object Registration with an Agent

Function	`search`
Ontology	`FIPA-Agent-Management`
Supported by	DF and AMS
Description	An agent may search for an object template in order to request information from an agent, in particular from a DF or an AMS. A successful search can return one or more agent descriptions that satisfy the search criteria and a null set is returned where no agent entries satisfy the search criteria. The DF or AMS description supplied must include a valid AID.
Domain	`object-description-template` ×[3] `search-constraints`
Range	Set of objects. In particular, a set of `df-agent-descriptions` (for the DF) and a set of `ams-agent-descriptions` (for the AMS).
Arity	2

6.2.4.1 Matching Criterion

The `search` action defined in this ontology mandates the implementation of the following matching criterion in order to determine the set of objects that satisfy the search criteria.

The first thing to note about the matching operation is that the `search` action receives, as its first argument, an object description that evaluates to a structured object that will be used as an object template during the execution of the `search` action. In the following explanation, the expressions *parameter template* and *value template* are used to denote a parameter of the object template, and the value of the parameter of the object template, respectively.

A registered object matches an object template if:

[3] Where × is Cartesian product.

1. The class name of the object (that is, the object type) is the same as the class name of the object description template, and,

2. Each parameter of the object template is matched by a parameter of the object description.

A parameter matches a parameter template if the parameter name is the same as the template parameter name, and its value matches the value template.

Since the value of a parameter is a term, the rules for a term to match another term template must be given. Before, it must be acknowledged that the values of the parameters of descriptions kept by the AMS or by the DF can only be either `SLConstants`, `SLSets`, `SLSequences` or other object descriptions (for example, a `service-description`).

The `search` action evaluates functional expressions before the object template is matched against the descriptions kept by the AMS or by the DF. This means that if the value of a parameter of an object description is a functional term (for example, `(plus 2 3)`), then what is seen by the matching process is the result of evaluating the functional term within the context of the receiving agent. A constant matches a constant template if they are equal.

Informally, a sequence matches a sequence template if the elements of the sequence template are matched by elements of the sequence appearing in the same order. Formally, the following recursive rules apply:

1. An empty sequence matches an empty sequence, and,

2. The sequence `(cons` *x* *`sequence1`*`)`[4] matches the sequence template `(cons` *y* *`sequence2`*`)` if:
   - *x* matches *y* and *`sequence1`* matches *`sequence2`*, or,
   - *`sequence1`* matches `(cons` *y* *`sequence2`*`)`.

Finally, a set matches a set template if each element of the set template is matched by an element of the set template. Notice that it is possible that the same element of the set matches more than one element of the set template.

6.2.4.2 Matching Example

The following DF agent description:

```
(df-agent-description
  :name
    (agent-identifier
      :name CameraProxy1@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :services (set
    (service-description
      :name description-delivery-1
      :type description-delivery
      :ontology (set Traffic-Surveillance-Domain)
      :properties (set
        (property
          :name camera-id
          :value camera1)
        (property
          :name baud-rate
          :value 1MHz)))
    (service-description
      :name agent-feedback-information-1
      :type agent-feedback-information
      :ontology (set traffic-surveillance-domain)
      :properties (set
        (property
```

[4] `cons` is the usual LISP function that it is here used to describe the semantics of the process. The function (which must not be considered part of the `FIPA-Agent-Management` ontology) takes two arguments, the second of which must be a list. It returns a list where the first argument has been inserted as the first element of its second argument. Example: `(cons` *x* `(sequence` *y* *z*`))` evaluates to `(sequence` *x* *y* *z*`)`.

```
          :name camera-id
          :value camera1))))
  :protocol (set FIPA-Request FIPA-Query)
  :ontology (set Traffic-Surveillance-Domain FIPA-Agent-Management)
  :language (set FIPA-SL))
```

will match the following DF agent description template:

```
(df-agent-description
  :services (set
    (service-description
      :type description-delivery
      :ontology (set Traffic-Surveillance-Domain)
      :properties (set
        (property
          :name camera-id
          :value camera1))
      :language (set FIPA-SL FIPA-SL1))
```

Notice that several parameters of the `df-agent-description` were omitted in the `df-agent-description` template. Furthermore, not all elements of set-valued parameters of the `df-agent-description` were specified and, when the elements of a set were themselves descriptions, the corresponding object description templates are also partial descriptions.

6.2.5 Retrieve an Agent Platform Description

Function	`get-description`
Ontology	`FIPA-Agent-Management`
Supported by	AMS
Description	An agent can make a query in order to request the platform profile of an AP from an AMS.
Domain	None
Range	`ap-description`
Arity	0

6.2.6 Terminate an Agent

Function	`quit`
Ontology	`FIPA-Agent-Management`
Supported by	All agents
Description	An AMS can ask an agent to terminate all execution on a given AP. Also, an agent can request the AMS to terminate the execution of an agent.
Domain	`agent-identifier`
Range	The execution of this function results in a change of state in the AMS but it has no explicit range set.
Arity	1

6.3 Exceptions

The normal pattern of interactions between application agents and management agents follow the form of the `FIPA-Request` interaction protocol (see [FIPA00026]). Under some circumstances, an exception can be generated, for example, when an AID that has been already registered is re-registered. These exceptions are represented as predicates that become true. This section describes all the predicates of the domain of discourse of the `FIPA-Agent-Management` ontology that represent exceptions of the interactions.

6.3.1 Exception Selection

The following rules are adopted to select the appropriate communicative act that will be returned in when a management action causes an exception:

- If the communicative act is not understood by the receiving agent, then the replied communicative act is `not-understood`.

- If the requested action is not supported by the receiving agent, then the communicative act is `refuse`.

- If the requested action is supported by the receiving agent but the sending agent is not authorised to request the function, then the communicative act is `refuse`.

- If the requested function is supported by the receiving agent and the client agent is authorised to request the function but the function is syntactically or semantically ill-specified, then the communicative act is `refuse`.

- In all the other cases the receiving agent sends to the sending agent a communicative act of type `agree`. Subsequently if any condition arises that prevents the receiving agent from successfully completing the requested function, then the communicative act is `failure`.

6.3.2 Exception Classes

There are four main classes or exceptions that can be generated in response to a management action request:

- `unsupported`: The communicative act and the content has been understood by the receiving agent, but it is not supported.

- `unrecognised`: The content has not been understood by the receiving agent.

- `unexpected`: The content has been understood by the receiving agent, but it includes something that was unexpected.

- `missing`: The content has been understood by the receiving agent, but something that was expected is missing.

6.3.3 Not Understood Exception Predicates

Communicative Act **Ontology**	`not-understood` `FIPA-Agent-Management`	
Predicate Symbol	**Arguments**	**Description**
`unsupported-act`	`String`	The receiving agent does not support the specific communicative act; the string identifies the unsupported communicative act.
`unexpected-act`	`String`	The receiving agent supports the specified communicative act, but it is out of context; the string identifies the unexpected communicative act.
`unsupported-value`	`String`	The receiving agent does not support the value of a message parameter; the string identifies the message parameter name.
`unrecognised-value`	`String`	The receiving agent cannot recognise the value of a message parameter; the string identifies the message parameter name.

6.3.4 Refusal Exception Propositions

Communicative Act	refuse
Ontology	FIPA-Agent-Management

Predicate symbol	**Arguments**	**Description**
unauthorised		The sending agent is not authorised to perform the function.
unsupported-function	String	The receiving agent does not support the function; the string identifies the unsupported function name.
missing-argument	String	A mandatory function argument is missing; the string identifies the missing function argument name.
unexpected-argument	String	A mandatory function argument is present which is not required; the string identifies the unrequired function argument.
unexpected-argument-count		The number of function arguments is incorrect.
missing-parameter	String String	A mandatory parameter is missing; the first string represents the object name and the second string represents the missing parameter name.
unexpected-parameter	String String	The receiving agent does not support the parameter; the first string represents the function name and the second string represents the unsupported parameter name.
unrecognised-parameter-value	String String	The receiving agent cannot recognise the value of a parameter; the first string represents the object name and the second string represents the parameter name of the unrecognised parameter value.

6.3.5 Failure Exception Propositions

Communicative Act	failure
Ontology	FIPA-Agent-Management

Predicate symbol	**Arguments**	**Description**
already-registered		The sending agent is already registered with the receiving agent.
not-registered		The sending agent is not registered with the receiving agent.
internal-error	String	An internal error occurred; the string identifies the internal error.

7 Agent Management Content Language

Agent Management uses `FIPA-SL0` as a content language which is defined in [FIPA00008].

8 References

[FIPA00008] FIPA SL Content Language Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00008/`

[FIPA00025] FIPA Interaction Protocol Library Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00025/`

[FIPA00026] FIPA Request Interaction Protocol Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00026/`

[FIPA00067] FIPA Agent Message Transport Service Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00067/`

[FIPA00079] FIPA Agent Software Integration Specification. Foundation for Intelligent Physical Agents, 2000. `http://www.fipa.org/specs/fipa00079/`

[RFC2396] Uniform Resource Identifiers: Generic Syntax. Request for Comments, 1992. `http://www.ietf.org/rfc/rfc2396.txt`

9 Informative Annex A — Dialogue Examples

1. The agent *dummy* is created and it registers with the AMS of its home AP:

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (ams-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :state active))))
```

2. The AMS agrees and then informs *dummy* of the successful execution of the action:

```
(agree
  :sender
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    ((action
      (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (ams-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :state active)))
    true))
```

```
(inform
  :sender
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (done
      (action
        (agent-identifier
        :name ams@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (ams-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :state active)))))
```

3. Next, *dummy* registers its services with the default DF of the AP:

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (register
        (df-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :protocol (set FIPA-Request Application-Protocol)
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type user-profiling
              :ontology (set meeting-scheduler)
              :properties (set
                (property
                  :name learning-algorithm
                  :value BBN)
                (property
                  :name max-nodes
                  :value 10000000))))))))
```

4. The AMS agrees and then informs *dummy* of the successful execution of the action:

```
(agree
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    ((action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc)
      (register
        (df-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :protocol (set FIPA-Request Application-Protocol)
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type user-profiling
              :ontology (set meeting-scheduler)
              :properties (set
                (property
                  :name learning-algorithm
                  :value BBN)
               (property
                 :name max-nodes
                 :value 10000000)))))))
    true))

(inform
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (done
      (action
        (agent-identifier
          :name df@foo.com
          :addresses (sequence iiop://foo.com/acc))
      (register
        (df-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :protocol (set FIPA-Request Application-Protocol)
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type user-profiling
              :ontology (set meeting-scheduler)
              :properties (set
                (property
                  :name learning-algorithm
                  :value BBN)
                (property
                  :name max-nodes
                  :value 10000000))))))))
```

5. Then, *dummy* searches with the DF for a list of meeting scheduler agents:

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (search
        (df-agent-description
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type meeting-scheduler-service)))
        (search-constraints
          :min-depth 2))))
```

```
(agree
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    ((action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (search
        (df-agent-description
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type meeting-scheduler-service))
        (search-constraint :max-depth 2))))
    true))

(inform
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (result
      (action
        (agent-identifier
          :name df@foo.com
          :addresses (sequence iiop://foo.com/acc))
      (search
        (df-agent-description
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type meeting-scheduler-service))
        (search-constraint :max-depth 2))))
        (set
          (df-agent-description
            :name
              (agent-identifier
                :name scheduler-agent@foo.com
                :addresses (sequence iiop://foo.com/acc))
            :ontology (set meeting-scheduler FIPA-Agent-Management)
            :languages (set FIPA-SL0 FIPA-SL1 KIF)
            :services (set
              (service-description
                :name profiling
                :type meeting-scheduler-service)
              (service-description
                :name profiling
                :type user-profiling-service))))))
```

6. Now *dummy* tries to modify the description of *scheduler-agent* with the DF, but the DF refuses because *dummy* is not authorised:

```
(request
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence (iiop://foo.com/acc))
      (modify
        (df-agent-description
          :name
            (agent-identifier
              :name scheduler-agent@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type meeting-scheduler-service))))))

(refuse
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    ((action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (modify
        (df-agent-description
          :name
            (agent-identifier
              :name scheduler-agent@foo.com
              :addresses (sequence iiop://foo.com/acc))
          :ontology (set meeting-scheduler)
          :language (set FIPA-SL0 KIF)
          :services (set
            (service-description
              :name profiling
              :type meeting-scheduler-service)))))
    (unauthorised)))
```

7. Finally, *dummy* tries to deregister its description with the DF, but the message is ill-formed and the DF does not understand (because the DF does not understand the `propose` performative):

```
(propose
  :sender
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (action
      (agent-identifier
        :name df@foo.com
        :addresses (sequence iiop://foo.com/acc))
      (deregister
        (df-agent-description
          :name
            (agent-identifier
              :name dummy@foo.com
              :addresses (sequence iiop://foo.com/acc))))))

(not-understood
  :sender
    (agent-identifier
      :name df@foo.com
      :addresses (sequence iiop://foo.com/acc))
  :receiver (set
    (agent-identifier
      :name dummy@foo.com
      :addresses (sequence iiop://foo.com/acc)))
  :language FIPA-SL0
  :protocol FIPA-Request
  :ontology FIPA-Agent-Management
  :content
    (propose
      :sender
        (agent-identifier
          :name dummy@foo.com
          :addresses (sequence iiop://foo.com/acc))
      :receiver (set
        (agent-identifier
          :name df@foo.com
          :addresses (sequence iiop://foo.com/acc)))
      :language FIPA-SL0
      :protocol FIPA-Request
      :ontology FIPA-Agent-Management
      :content
        (action
          (agent-identifier
            :name df@foo.com
            :addresses (sequence iiop://foo.com/acc))
          (deregister
            (df-agent-description
              :name
                (agent-identifier
                  :name dummy@foo.com
                  :addresses (sequence iiop://foo.com/acc))))))
    (unsupported-act propose)))
```

10 Informative Annex D — ChangeLog

10.1 2001/10/03 - version H by FIPA Architecture Board

Page 24, line 825: Changed incorrect reference of AMS to DF.