Evaluation of FIPA-OS 1.03

Mikko Laukkanen

Helsinki, 16th February 2000

Cellular System Development,
Sonera Mobile Operator,
Sonera Ltd,
P.O.Box 970, 00051 SONERA
mikko.laukkanen@sonera.com

Abstract

FIPA-OS is an open-source implementation of the mandatory elements of FIPA 97-compliant agent platform. The primary aim of FIPA-OS is to provide a framework supplementing the FIPA specifications and therefore make it easier to adopt the FIPA technology. In this paper, FIPA-OS is evaluated according to following criteria: FIPA-compliance, software quality, interoperability and scalability. The objective of this paper is to give feedback to the developers of FIPA-OS. The results show that interoperability is the area needing improvements. This paper is based on author's Master's Thesis "Evaluation of FIPA-compliant agent platforms", which is available on request.

Keywords: FIPA-OS, agent platform, evaluation, scalability, interoperability

Contents

1 Introduction

Agent based software and agent systems vary making them difficult to interoperate. FIPA (Foundation for Intelligent Physical Agents) is a standardization organization promoting development and specification of agent technologies. The main goal for FIPA is to specify, how different kinds of agent platforms can interoperate. FIPA specifications have reached such a maturity that FIPA-compliant agent platforms have been implemented by various developers.

In this paper, FIPA-compliant agent platform called FIPA-OS from Nortel Networks is evaluated. Evaluation is based on our experiments on using FIPA-OS. The criteria for evaluation are: FIPA-compliance, software quality, interoperability and scalability. This paper is based on author's Master's Thesis "Evaluation of FIPA-compliant agent platforms" [3], which is available on request. Although in the Master Thesis there was version 1.02 of FIPA-OS studied, in this paper, we take the version 1.03 into account.

Objective of this paper is to give feedback to the developers of FIPA-OS and to gain our own knowledge on FIPA-compliant agent platforms when deciding, which platform to use in our own extensions and applications.

This paper is organized as follows. Section 2 presents the criteria used in evaluation and describes, how tests and measurements are carried out. In section 3 the results are presented and justified. Section 4 discusses the results and concludes this paper.

2 Evaluation criteria

2.1 Requirements for a FIPA-compliant agent platform

For the evaluation of the FIPA-compliance we developed a requirement specification for a FIPA-compliant agent platform. The requirement specification is based on the FIPA specifications, and basically collects the requirements from informal specifications into a formal UML-compliant [1] requirement specification. The motivations for collecting these requirements were:

- Lack of a requirement specification document of a FIPA-compliant agent platform,

- The need to define, what the "FIPA-compliance" actually means and what is needed to achieve the "FIPA-compliance", and
- Our intention to come up with a UML-specifications for the requirements serving as a reference in the evaluation of the different platforms.

From the requirement specification the most important requirements were used in evaluation. The most important requirements include interoperability, Agent Management System (AMS), Agent Communication Channel (ACC) and Directory Facilitator (DF). These requirements are collected in a table in Appendix 1. In addition to the requirements, a set of optional features are collected in Appendix 2. The requirements are the ones that every FIPA-compliant agent platform must fulfill, whereas the optional features are just for bringing in extra value to a platform. The evaluation for FIPA-compliance is based on the requirements so that the requirements are gone through one by one. Requirements and optional features are dealt separately and from both groups, the percentage of fulfilled requirements is calculated.

2.2 Software quality metrics

In addition to the requirements, more traditional metrics for software quality were also used in evaluation. Software quality was evaluated, because we wanted to find out, how suitable FIPA-OS is for our further development and extensions, so the main emphasis is put on the availability of sources, design and implementation issues and possibilities for own further extensions. Software quality metrics are presented in Table 1.

The first column implies the criterion. Explanations for the criteria are collected in Table 2.

All the criteria are evaluated in the range of 0 to 5. The "Factor" column indicates, how much emphasis is put on the criterion. The final column indicates the actual scale of the criterion after the "grade" of the criterion is weighted with the factor. The scale s_c for criterion c is derived as:

$$s_c = [0, 5 * f_c], \tag{1}$$

where f_c is the weight factor for c.

Table 1: Software quality metrics for a FIPA-compliant agent platform.

Criterion	**Factor *f***	**Scale *s***
User interaction	0.2	0 - 1
Documentation		0 - 4
- Programmer's guide	0.8	0 - 4
- User's guide	0.8	0 - 4
- Installation guide	1.0	0 - 5
- JavaDoc	0.6	0 - 3
Ease of installation	0.6	0 - 3
Availability of binaries	1.0	0 / 5
Availability of sources	0.8	0 / 4
Technical merits		0 - 4.6
- Architecture	1.0	0 - 5
- Design	1.0	0 - 5
- Implementation	0.8	0 - 4
Ease of implementing agents	0.8	0 - 4
Further development	1.0	0 - 5

So if the ease of implementing agents is evaluated and grade of 3 is assigned to it, the actual grade after weighting is $0.8 * 3 = 3.2$, while the maximum would be $0.8 * 5 = 4$.

The documentation and technical merits are divided into sub-criteria. In these cases the total range s_{total} is calculated as the mean value of the weighted sub-criteria s_{sub} as:

$$s_{total} = \frac{\sum_1^n s_{sub}}{n}, \tag{2}$$

where s_{sub} is the scale for sub-criterion and
n is the number of sub-criteria.

The idea of weighting the criteria is from author. The evaluation is based on author's experiences in using FIPA-OS and implementing agents with it.

2.3 Interoperability

Interoperability is achieved by using IIOP as the baseline protocol and Agent Communication Language (ACL) as the communication language. In the evaluation of interoperability, the implementation of IIOP was inspected. To support our inspections, we ran some cross-platform tests against JADE, which is a FIPA-compliant agent platform from CSELT.

Table 2: Explanations for the criteria in software quality.

Criterion	**Explanation**
User interaction	What kind interfaces FIPA-OS provides for both developer and user, e.g. GUIs, debug/log information and ease of configuration.
Documentation	Availability of documents and guides.
Ease of installation	How much effort is needed to get FIPA-OS installed and running.
Availability of binaries	Self explanatory.
Availability of sources	Self explanatory.
Technical merits	Design and implementation issues, reusability of the code, organisation of the sources and binaries.
Ease of implementing agents	How much efforts are needed to implement a custom agent in FIPA-OS.
Further development	How easy it is to make own extensions to the platform, how much does a developer need to make changes to the original code.

2.4 Stress tests

Stress tests were performed for three reasons:

- To evaluate the scalability of FIPA-OS.
- To compare the performance of IPMT to other transports.
- To give an overall view to performance of different transport mechanisms.

The evaluation of scalability is the main objective of the stress tests. A test agent, which sends a message to another agent and gets back the response, was implemented. The time between message sending and arrival of response was measured. To see how FIPA-OS can handle big amounts of simultaneous requests, 1, 50, 100, 150 and 200 test agents were used in sending simultaneous messages. These tests were repeated five times and mean values and standard deviations of the data points were calculated.

The stress tests were performed on Linux systems. The properties of the testing environment are collected in Appendix 3.

Stress tests also helped to see, how IPMT performed compared to IIOP. Because IPMT can and is usually implemented as direct method calls, IPMT is supposed to perform a lot faster than IIOP messaging. It must be noted though that this applies only to a situation, where a platform resides in one host machine. If the platform is distributed across multiple host machines, direct method calls cannot be used. By testing and comparing the performances of both IPMT and IIOP it can be seen, if the gap in performance is so big that the implementation of specific IPMT is worthwhile. After all, ORBs should be able to detect if the method call is local or not. If the receiver of the call is local object, the call will not be forwarded to the network protocol stack and therefore the method will be executed as local method call.

Instead of putting the emphasis on response times and possibly comparing the performances to other platforms, we think that it is more important to see, how FIPA-OS and its message transports react to a congested situation. Therefore, when inspecting the results, a special attention should be paid to the overall scalability. It should be also noted that the tests measure response times from one agent's point of view, when it is sending one request to a platform under heavy load. The test measuring response times, when one agent sends a multicast message, is left for further study. Also, our tests were conducted using zero-size payload[1] in the ACL-messages. Thus, tests with variable sized payload are left for further study.

3 Results

3.1 FIPA-compliance

The result on FIPA-compliance are presented in Table 3. Requirement R14 was not fulfilled, because in FIPA-OS, an agent is able to send a message to another agent without resistering to AMS. R18 was the only one, which was a little questionable. While FIPA-OS 1.03 provides the use of multiple transports, we were not able to verify, if ACC is really able to choose the correct transport according to the address format of the receiver-field in the ACL-message. R2 causes also troubles in FIPA-OS, because the IDL is not exactly as it is specified in FIPA 97. This issue is discussed later in section 3.3.

In optional features, F1, F5, F8, F10, F12 and F15 were not supported. F1 states that other protocols may be supported, and FIPA-OS does this, but the initial contact must be made using the baseline protocol. In FIPA-OS, user is able to use other than the baseline protocols

[1] In ACL this means that the `:content`-field is empty.

explicitly. F5 and F8 are not supported, because FIPA-OS does not support mobile agents. F10 and F12 are not supported by FIPA-OS itself, although an explicit registration can be made by the user. F15 states that DF may have some criteria for restricting services. In FIPA-OS, this is not supported.

One remark should be made about the functionalities of AMS and DF. In FIPA-OS 1.03, the GUI for DF is separated from the DF itself, which makes the DF-GUI an agent of its own. This is fine. However, AMS has no GUI for controlling agent lifecycles. It would be nice to have this kind of GUI for AMS also.

Table 3: Requirements and optional features.

Requirements fulfilled	%	**Optional features fulfilled**	%
27/28	96%	9/15	60%

3.2 Software quality

The results on software quality are presented in Table 4. In the table, the weighted results are summed and the total is presented at the bottom line with the percentage of the maximum.

GUIs have improved in version 1.03. FIPA-OS provides a test agent, which can be used in testing messaging using a GUI for inputting and sending an ACL-message. This GUI can also be used by any other agent. AMS and ACC do not provide any GUIs, but DF does by showing the contents of its directory. For the developer FIPA-OS provides an easy way of producing debug and log information by exploiting the Metamata diagnostics-package. However, in FIPA-OS 1.03, Metamata-package is not needed anymore, because it is replaced by a similar package provided by FIPA-OS itself.

Documentation in FIPA-OS needs improvements. Only a document containing distribution notes is delivered with FIPA-OS. While it contains sufficient instructions for installation, a programmer's guide and some example agents should be definitely included. Currently FIPA-OS includes only one example agent, which can be used to test message sending/receiving and which we used as a reference in the implementation of our own test agent.

If the need for downloading and installation of third-party tools is not taken into account, FIPA-OS is easy to install; there are scripts for both Windows NT and Unix systems provided

Table 4: Results on software quality.

Criterion	**FIPA-OS**
User Interaction	0.8
Documentation - Programmer's Guide - User's Guide - Installation Guide - JavaDoc	1.6 0 2.4 4 0
Ease of installation	0.6
Availability of binaries	5
Availability of sources	4
Technical merits - Architecture - Design - Implementation	3.8 5 4 2.4
Ease of implementing agents	3.2
Further development	5
Total Total %	24 78%

for compiling and running the platform.

What it comes to technical merits, FIPA-OS is organised logically. Every component is collected into a package. FIPA-OS provides its own variation of Makefile and scripts to execute it. We think though that it would be better off to use standard Makefiles, which would make FIPA-OS easier to port on different operating systems and would allow a selective compilation of individual source files in any directory.

Unlike other platforms, FIPA-OS supports platform and agent profiles, which are encoded in XML/RDF [6]. In addition to agent profiles, XML can be used as a content language in ACL-messages (this feature was not tested though). The use of XML is justified for many reasons. Firstly, XML is a standard for expressing and exchanging information on the Web [2]. Secondly, syntax validation of the content is possible, when a DTD has been defined. In ACL, the DTD is defined in `:ontology`-field. Thirdly, XML documents are easy to convert into a human-readable form (e.g. Web page) so that their content can be stored in this form for instance into log files, where end-users can easily check the contents. Fourthly, there exist many tools for parsing, editing and translating XML documents. In our experiment, where we generated a number of test agents, we would have need for a default profile. In FIPA-OS, the

profile name is constructed by the agent's name, and because the names for test agents were generated randomly, the functionality for a default profile had to be implemented on our own.

FIPA-OS is an open-source platform. Especially for this reason, we think that the code should follow a coding style, which would make the code much more readable and understandable. This is important especially if the development of FIPA-OS is going to be carried out by other people than the ones working for Nortel Networks. All the people making contributions would be enforced to follow the FIPA-OS specific coding style. We think that this would make the maintainability of the source code much easier.

Implementing an agent in FIPA-OS is fairly easy. However, because of the absence of class and functional descriptions, a user has to use existing agents as reference. Also, there are some things that should be transparent to the user, such as registration with AMS. It should not be a user's responsibility to manually construct an ACL-string for the AMS-registration message. These kinds of things should be implemented in superclasses and a method to do the actual job should be provided for the user.

FIPA-OS is easy to extend. With well-organised and component-oriented structure, a user wanting to make an extension to the platform usually needs only to implement a new class or a package for the extension. There is no need to change the existing code. However, openess of the source code makes it possible to make modifications to the original code, if needed.

3.3 Interoperability

FIPA-OS provides two possible implementations for IIOP: Voyager 3.11 and JDK1.2 ORB. In previous versions of FIPA-OS, OrbixWeb from Iona was supported, but now that FIPA-OS uses JDK1.2, the OrbixWeb ORB is replaced by the JDK's own ORB. In addition to IIOP-transports, Java RMI is supported.

FIPA-OS also presents a solution for bootstrapping problem. Bootstrapping problem occurs, when a platform is started up and it does not know the addresses of other platforms. In FIPA-OS, the URL- and IOR-addresses are published in a Web-server, from where they are also queried. While the idea is excellent, the implementation is a bit weak; when taking a deeper look at the implementation of the lookup and resolving functions, we found out that the IOR was not retrieved from the Web-server. Instead, the ORB's own methods for constructing IOR was used and the CORBA naming service was used in object reference lookups. We see no

reason for using CORBA naming service, because the IORs are retrieved from the Web servers. Using naming service may result in interoperability problems among different ORBs.

The implementation of IIOP and the FIPA_Agent_97-IDL contained an error, which made interoperability fail with JADE. The reason for this is that in FIPA-OS 1.03 the IDL-interface is wrapped inside a CORBA-module, which is not in line with FIPA 97 specifications. In earlier versions of FIPA-OS, the IDL was implemented as it is specified by FIPA. Why has this been changed?

However, in general, the main problem in interoperability is that the use of URL- and IOR-addresses varies among FIPA-compliant agent platforms. While some platforms support only IOR-addresses, the others rely on using URL-addresses. FIPA-OS supports URL-addresses but provides a way of publish IORs in a Web-server. We think that this is the right way to do it. IORs should be used in addressing and URL-format can be used in looking up the IOR.

In our opinion, the use of IIOP as the baseline protocol is a right decision, because IIOP and CORBA have proved to be successful in object distribution. Also, OMG [5] is constantly making efforts in developing CORBA standard. However, we think that the specification for interoperability needs some improvements by FIPA. Especially the mapping between URL-addresses and IOR-addresses should be specified more precisely or examples of doing it should be provided. In the current specification an address is composed of host name (or IP-address), port number and object key. First problem in this kind of addressing is the object key. Unfortunately, most of the CORBA ORB implementations available do not allow users to choose the object key. Instead, ORBs create some arbitrarily value, which may even be binary characters. Second problem is the port number, which may not be predefined. On the contrary, ORBs tend to choose an available port number from some predefined range.

3.4 Scalability

In this section, the results on stress tests are presented. The results are collected in Table 5. In the table there are mean values ($\bar{x}$) and standard deviations (σ) presented. The first column indicates the number of simultaneous messages sent. The remaining columns indicate the means and standard deviations of different transports. All the measurements are milliseconds. The results are plotted in Figure 1 for easier interpretation.

FIPA-OS does not implement any specific IPMT. Instead, all the communication is gone

Table 5: The results for scalability.

	IPMT		**IIOP**	
Msg #	$\bar{x}$	σ	$\bar{x}$	σ
1	221	$\pm$ 3	239	$\pm$ 7
50	1363	$\pm$ 309	1602	$\pm$ 336
100	2236	$\pm$ 1203	4317	$\pm$ 1023
150	4496	$\pm$ 1766	5689	$\pm$ 2254
200	6214	$\pm$ 2208	6335	$\pm$ 2120

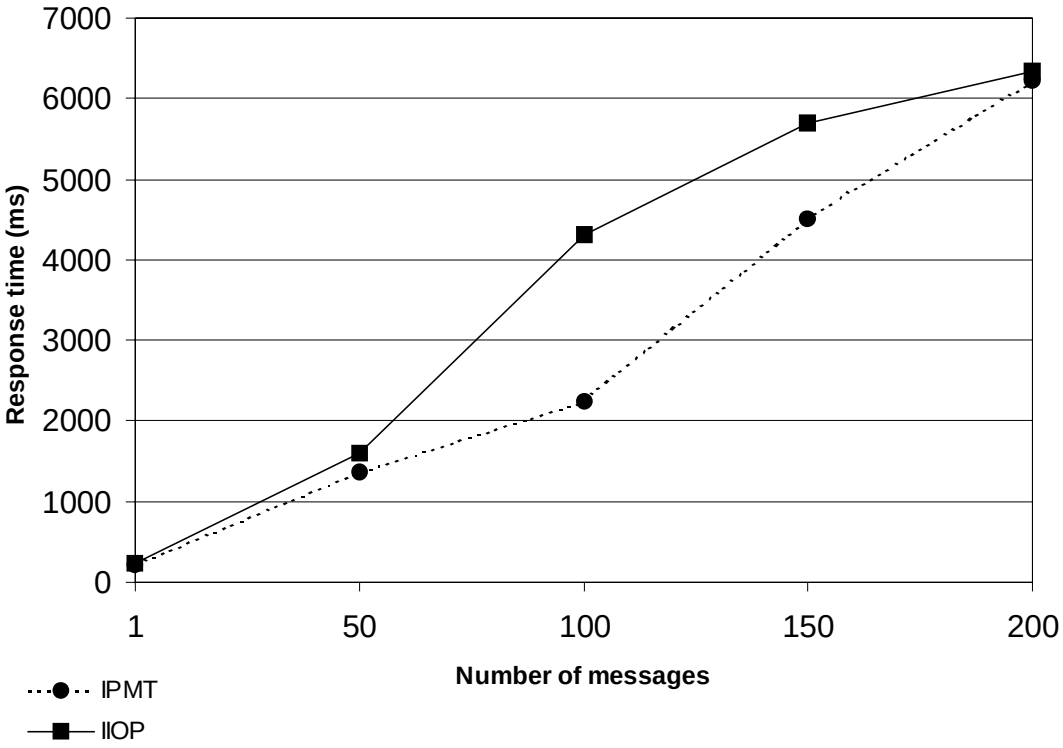

Figure 1: FIPA-OS results.

through CORBA-ORB. Therefore, FIPA-OS counts on the ORB's performance in intra-communication. According to our experiments, this worked out well with scalability. In fact, current ORB's should be able to detect, if the receiving end for the message resides in the same host, and if it does, the ORB does not forward the message to the network protocol stacks. Instead, the message is delivered using a local method call. However, results show that there is not a big gap in response times between IPMT and IIOP. In JADE, where IPMT is implemented using local method calls, the difference in performance is dramatic.The approach FIPA-OS has chosen is safe in a way that the distribution of the platform across multiple host machines does not introduce any probles. On the other hand, if an application places high demands for performance in agent communication, FIPA-OS should provide a faster (and optional) way of carrying out IPMT. We think that it is rare to have a platform distributed across multiple

machines. Therefore we propose that local method calls would be used in IPMT, at least as an optional feature.

As we ran our tests under Linux system, we faced some errors either in FIPA-OS or Java Virtual machine implementation. First we used JDK1.2.1, which frequently seized communication with more than 50 messages. As we updated to JDK1.2.2, this error did not occur anymore, but with 100 or more messages, Java Virtual Machine crashed occasionally. We are not sure about the reason for this, but we believe it has something to do with the thread allocation and scheduling of the Linux-port of Java Virtual Machine.

The results show that the standard deviations are big especially with big loads making it hard to know, how much confidence we can place in the results. Because of this, we established confidence levels of 0.01 and therefore could see, what the response time would be in 99 out of 100 requests. Although in the literature it is said that a confidence level of 0.05 is usually sufficient [4], we dicided to use a confidence level of 0.01. The response times are shown in Table 6 in milliseconds.

Table 6: Response times with confidence of 0.01.

Msg #	**IPMT**	**IIOP**
1	228	256
50	1464	1712
100	2513	4553
150	4828	6113
200	6574	6681

4 Discussion and conclusions

FIPA-OS fullfilled our requirements for a FIPA-compliant agent platform well. 60% of the optional features are supported, but while FIPA-OS evolves, these features are likely to be supported in future releases. Also, because of the openess, the errors in the code can be found quickly.

FIPA-OS exploits many third-party tools, such as XML-parsers, ObjectSpace Voyager and Apache Web-server. The idea of using third-party tools is good, but these tools should be included in FIPA-OS, or at least there should be a distribution package available, where all these tools are included and their versions freezed. Many companies tend to have only the latest

versions of their products available, and if FIPA-OS uses some older versions of that product, the end-user is in trouble with finding the correct versions. Furthermore, the installation of these tools should be integrated into the installation of FIPA-OS.

While the the quality of the software is good, although a coding style would be needed, the documentation needs improvements. In our opinion, being an open-source distribution does not justificate the absence of decent documentation. A class diagram of the whole architecture, textual descriptions of the classes and their methods as well as examples of using them should be included.

Interoperability did not work out because of the incorrect FIPA_Agent_97-interface. In the earlier versions of FIPA-OS, where the IDL was correct, interoperability with JADE was achieved. The idea of using a Web-server in publishing IORs is good and is in line with our opinion of the correct implementation of inter-platform messaging. FIPA-OS and its transports are also well scalable.

FIPA-OS is still under development, but has already proved to be a suitable platform especially for extensions and "user-customized" agent platform. As distributed under open-source license makes this possible. As for being a "black-box" middleware for agent and agent system application development, FIPA-OS has to improve in documentation and in correcting some errors it contains.

As a conclusion, the pros and cons of FIPA-OS are presented below:

Pros:

+ Free and open source.
+ Fulfills the requirements for a FIPA-compliant agent platform well.
+ Well-organised and logical structure of the distribution.
+ Fast, easily extendable and scalable transports.
+ Agent profiles and content language as XML/RDF.
+ Distribution of IORs using Web-server.
+ JavaDoc-documentation.
+ Use of third party tools.
+ GUIs for user interaction.

Cons:

- The need of downloading and installing third-party tools.
- Installation is complex because of the third-party tools.
- Lack of standard Makefiles.
- Lack of example agents and coding style.
- Lack of programmer's guide, class diagrams and functional descriptions.
- Interoperability problem due to FIPA_Agent_97-IDL interface.

References

[1] Booch Grady, Rumbaugh James, and Jacobson Ivar. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] FIPA. FIPA 98 Specification, Spec 13, Version 1.0, FIPA97 Developers Guide, 1998.

[3] Laukkanen Mikko. Master's Thesis: Evaluation of FIPA-compliant agent platforms, 1999.

[4] Newman William M., and Lamming Michael G. *Interactive System Design*. Addison-Wesley, 1996.

[5] OMG. OMG homepage. In *http://www.omg.org/* (1999).

[6] W3C. Extensible Markup Language (XML) 1.0 - W3C Recommendation. In *http://www.w3.org/TR/1998/REC-xml-19980210* (Feb 1998).

APPENDICES

Appendix 1. The mandatory requirements for a FIPA-compliant agent platform.

No	Requirement	Source
	Interoperability	
R1	IIOP must be used as the baseline protocol.	97
R2	Baseline protocol must define an IDL-interface called FIPA_Agent_97 containing one method called "message", which takes as an input parameter a CORBA string.	97
R3	ACL must be used in inter-platform communication.	97
R4	Address in the form of URL must be supported.	97
	FIPA agent	
R5	FIPA agent must be able to communicate with other FIPA agents.	97
R6	FIPA agent must have an unique identity.	97
R7	FIPA agent has one to many addresses.	97
R8	FIPA agent must have a home platform, which is static.	97
R9	FIPA agent's name is associated with the agent when the agent is created or when it registers with AMS for the first time.	97
R10	FIPA agent's address must not be used as agent's name.	98
R11	FIPA agent must have one or more owners.	97
	ACL and messages	
R12	FIPA agent must be able to send a "not-understood"-message, if the incoming message is either syntactically or semantically malformed or is not supported by the agent.	97
	AMS	
R13	AMS controls and administrates the lifecycles of the FIPA agents registered with it.	97
R14	FIPA agent must be registered with AMS in order to interact with agents in the same platform or agents residing in other platforms.	97
R15	AMS maintains indexes to the agents registered with it (acts as "white pages" of the platform).	97
R16	Only one AMS administrates a platform.	97

Appendix 1. Cont'd.

No	Requirement	Source
	ACC	
R17	ACC must support IIOP.	97
R18	ACC must be able to choose the correct transport protocol according to the address format.	97
R19	ACC must take care of checking the syntax of incoming message before forwarding it onwards.	97
R20	Agents need not to be aware of the details of the message transports.	97
R21	ACC must support singlecast message transport.	97
R22	All messages are sent via either local or remote ACC.	98
R23	ACC must support both intra- and inter-platform communication.	97
R24	Only messages addressed to agent(s) are sent via ACC.	97
R25	ACC needs not to be an agent. Instead, ACC just provides message transport services.	98
R26	In message routing, ACC does not access the content of the messages.	97
	DF	
R27	Every platform must include at least one DF.	97
R28	If DF is accessed remotely (i.e. from another platform), then the DF must provide an ACL interface.	97

Appendix 2. The optional features of a FIPA-compliant agent platform.

No	Feature	Source
	Interoperability	
F1	Other protocols than IIOP may be supported, although the initial contact must be done using IIOP.	97
F2	An agent may support IIOP by itself.	97
F3	Address in the form of IOR may be supported.	98
	FIPA agent	
F4	FIPA agent may be registered with a platform (i.e. be resident) or it can be non-resident.	98
F5	FIPA agent may have a foreign platform, which changes while the agent moves.	98
	ACL and messages	
F6	SL may be supported as content language.	97
F7	XML may be supported as content language.	97
	Mobility	
F8	FIPA agent may move from a location to another.	98
	AMS	
F9	A platform may extend across multiple machines.	97
F10	AMS may advertise itself with DF.	98
	ACC	
F11	ACC may support other than IIOP.	97
F12	ACC may advertise its transports with DF.	98
	DF	
F13	DF may be distributable.	97
F14	DF may register itself with other DFs.	97
F15	Services offered by DF may be restricted by some criteria.	97

Appendix 3. Test environments

```
REQUESTING PLATFORM
==================

LINUX SYSTEM INFORMATION
------------------------
Linux version 2.2.12-20 (root@porky.devel.redhat.com)
                        (gcc version egcs-2.91.66 19990314/Linux
                        (egcs-1.1.2 release)) #1 Mon Sep 27 10:40:35 EDT 1999
RedHat 6.1
GNOME-window manager

CPU INFORMATION
---------------
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 7
model name      : Pentium III (Katmai)
stepping        : 3
cpu MHz         : 497.841506
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
sep_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge
                  mca cmov pat pse36 mmx osfxsr kni
bogomips        : 496.44

MEMORY INFORMATION
------------------
        total:    used:    free:    shared:   buffers:  cached:
Mem:  263909376 244039680 19869696 183885824  5955584  160497664
Swap: 549601280 27820032  521781248
MemTotal:    257724 kB
MemFree:      19404 kB
MemShared:   179576 kB
Buffers:       5816 kB
Cached:      156736 kB
SwapTotal:   536720 kB
SwapFree:    509552 kB

NETWORK
-------
10Mb/s LAN (closed)
```

Appendix 3. cont'd...

```
SERVER PLATFORM
===============

LINUX SYSTEM INFORMATION
------------------------
Linux version 2.2.5-15smp (root@porky.devel.redhat.com)
                          (gcc version egcs-2.91.66 19990314/Linux
                          (egcs-1.1.2 release))
                          #1 SMP Mon Apr 19 22:43:28 EDT 1999
RedHat 6.1
KDE-window manager

CPU INFORMATION
---------------
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 5
model name      : Pentium II (Deschutes)
stepping        : 2
cpu MHz         : 398.779465
cache size      : 512 KB
fdiv_bug        : no
hlt_bug         : no
sep_bug         : no
f00f_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
                  pge mca cmov pat pse36 mmx osfxsr
bogomips        : 398.13

MEMORY INFORMATION
------------------
        total:    used:    free:   shared:   buffers: cached:
Mem:  529707008 42729472 486977536 32075776  4788224  25747456
Swap: 707330048        0 707330048
MemTotal:    517292 kB
MemFree:     475564 kB
MemShared:    31324 kB
Buffers:       4676 kB
Cached:       25144 kB
SwapTotal:   690752 kB
SwapFree:    690752 kB

NETWORK
-------
10Mb/s LAN (closed)
```