COMMUNICATION AND MANAGEMENT EXPERICENCES IN AN E-COMMERCE MAS-BASED ENVIRONMENT

Francisco Valera [1], Jorge E. López·de·Vergara [2], José I. Moreno[1], Víctor A. Villagrá [2], Julio Berrocal [2]

[1] *Area de Ingeniería Telemática, Universidad Carlos III de Madrid*
Avda. de la Universidad 30, 28911 Leganés (MADRID)
Email: fvalera@it.uc3m.es, jmoreno@it.uc3m.es
[2] *Departamento de Ingeniería Telemática, Universidad Politécnica de Madrid*
Ciudad Universitaria s/n, 28040 MADRID
Email: jlopez@dit.upm.es , villagra@dit.upm.es, berrocal@dit.upm.es

1. Introduction

Electronic Commerce is based on an efficient and precise information interchange between participating stakeholders (typically customer and provider), using the existing communication infrastructure. Internet and intranets communication facilities are making the amount of available information quite large and it keeps on increasing progressively (at the same time, information requested by users, is gradually rising too).

It is imperative to find some mechanism that allows clients and servers to interchange information in a fluent and precise manner, minimizing the time normally spent in finding really useful information, that tends to be quite high.

This article describes a case of study of a multi-agent system (*MAS*) based intermediation service for e-commerce, implemented with Java and CORBA. After an architectural overview of the different components (sections 2 and 3), the last two sections will mainly focus on developing communication and management system experiences.

1.1 Brokerage service

Brokerage service (provided by an information broker) is a very helpful tool to promote symmetry in the context of e-commerce, allowing customers and providers, to communicate in a simpler manner. Obviously, the main problem it has to face, is information acquisition and subsequent information delivery specifically required by users.

Generically, this brokerage service enables navigation through an information index, so that users can begin a search procedure in order to find what is demanded.

There are however more flexible and transparent approaches, based on personal recommendations (user profiles), with the intention that users are not forced to go into the navigation process, which sometimes may become a little tedious [1, 2]. We will distinguish between two of these systems:

- *Natural-language approach*: users can make requests and select the desired information more comfortably. They can also improve their profiles and the searching mechanism increasingly, evaluating the received results.
- *Collaborative entities*: the main idea is to create a common knowledge, so that users can get information from each others and mutually take advantage of this learning

mechanism, while looking for useful information. Agent technology will be very relevant here [1].

1.2 Mediation

A brokerage service of the kind we have introduced, would be considered as a constitutive part of what is usually known as mediation platform [3].

This platform includes the set of duties that are supposed to rule every transaction associated with e-commerce: *brokerage, security, delivery, accounting, billing and connectivity.* All these tasks are oriented to decrease the distance between users and providers from a commercial viewpoint.

2. ABROSE PROTOTYPE

Following all these ideas, an implementation of an electronic brokerage service based on agent technology, Java (*JDK1.1.7+Swing*) and CORBA (*OrbixWeb3.1*), with a fully functional management system, has been developed in the context of ABROSE[*] project (*Agent Based Brokerage Services in Electronic Commerce*), belonging to the European Commission ACTS program.

ABROSE, formally specified using UML (with *Rational™ Rose98™*), explicitly emphasizes the brokerage task (enforced by agent technology), including the two approaches we mentioned (navigation and collaboration).

2.1 Objectives

The main proposed objectives in ABROSE are:

- Dynamic knowledge capture.
- Usage of a multi-agent system to represent the knowledge base.
- Navigation and information retrieval through a graphical interface (also supported by agent technologies). It will also help (together with agents) providers in their labour of registering and propagating their offers.
- Utilization of Java and CORBA as implementation technologies.
- Usage of collaborative agents to optimise intermediation.
- Evaluate agent technology implications in information brokerage and commercial applications.
- Java, CORBA, SNMP usage to implement the management architecture.

2.2 Functionality

From a functional viewpoint, ABROSE system is an Internet accessible brokerage service that covers the commented lacks and faces the diverse technologic objectives we have just listed.

[*] In this project ACTS ABROSE (AC-316), the participant entities are: T-Nova (DT-Berkom), CNET (France-Telecom), Univ. of Toulouse, National Technical Univ. of Athens, Univ. of Berlin, Infomures (Romania), Tradezone Ltd. England, Dégriftour France, Univ. Politécnica de Madrid, Univ. Carlos III de Madrid

The user accesses the system from a Web browser (with the subsequent access phase), and once inside the service, a graphical interface appears and the user is able to ask the broker to collect some information or navigate through its knowledge base (figure 1).

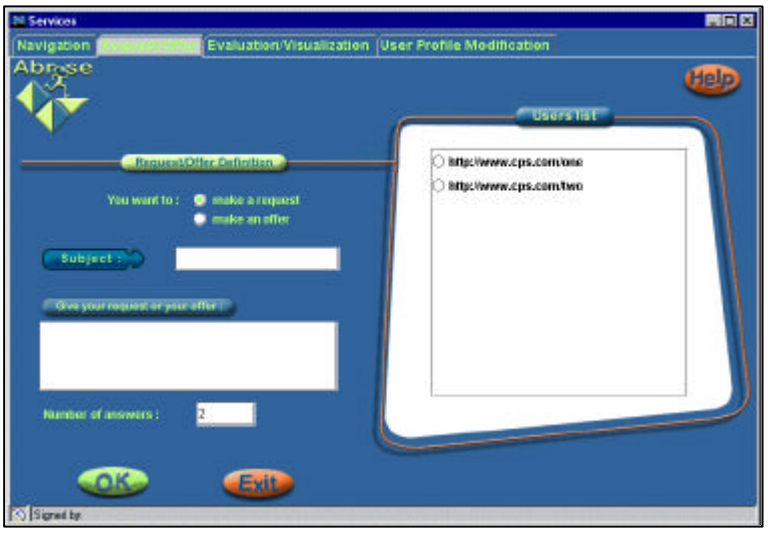

Figure 1. *Request Interface*

Contrasting with an implementation where the broker would consult the different providers and give the answers back, directly represented in the browser [7,11], in ABROSE we have the broker exhaustively searching for providers related to the request and users having to access the providers by themselves.

Agents technology allows the broker to learn users preferences transparently, increasing its accuracy when building this related providers list we have talked about.

2.3 Architecture

System architecture is based on two well differentiated domains: the *broker domain*, which has the dual mission of brokerage and system management and the *user domain*, that assumes all the functionality of displaying information to the end-user (navigations, requests, answers, etc.).

Figure 2 shows the system architecture with the different blocks and relations between them. Some management and communications related blocks have been omitted and will be further developed later.

The rest of this section covers the different modules, classifying them in the described domains. This description will allow a much more precise understanding of the system functioning.

User Domain

- *Connection Assistant* (CA): helps users to connect and join the system.
- *Agent Management Assistant* (AMA): coordinates intra-domain and inter-domain communications.

- *Navigation Assistant* (NA): allows users (providers or customers) to navigate through the broker knowledge base and to select relevant domains and criteria in order to improve their queries.
- *Spy*: gets information from user requests to enhance user's preferences from the broker viewpoint (user profile), so that precise information acquisition will be easier.
- *Front End* (FE): access point to provider information. The user will get a reference to connect FE as the answer to his requests.

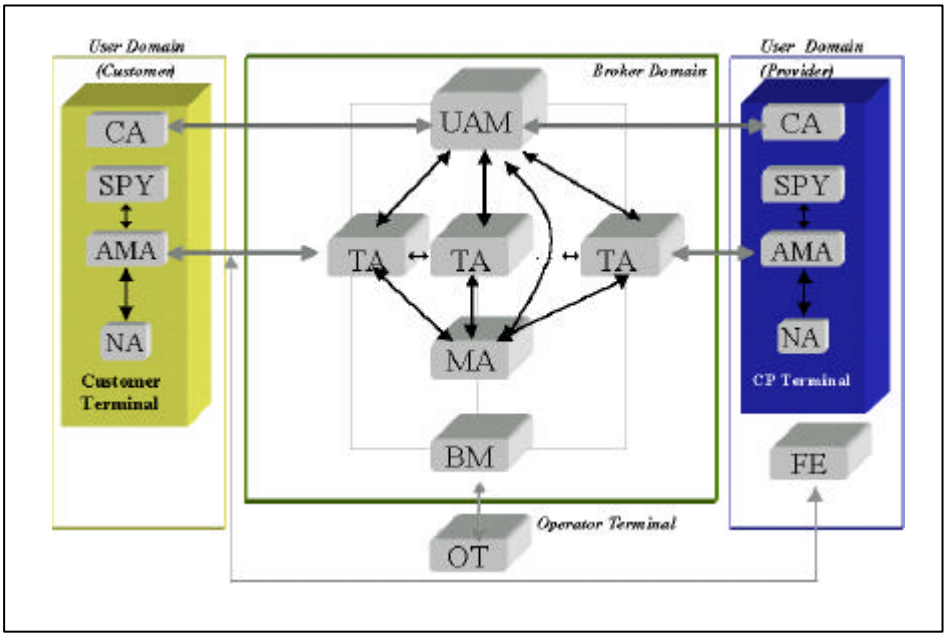

Figure 2. *ABROSE architecture*

Broker Domain

- *User Access Manager* (UAM): manages user profiles and verifies the access to ABROSE. The user profile has for example, information to perform the access phase (login and password). It is created at system initialisation and managed while the service is working.
- *Broker Manager* (BM): monitories interactions between various modules of the system. It is also responsible for system initialisation. Management architecture will be explained in section 5.
- *Operator Terminal* (OT): helps the system operator to perform management and administration of the service provided by ABROSE. It shows modules status, making access to management parameters easier.
- *Multi-Agent System* (MAS): in the broker domain, agents represent both clients and providers. Every agent is associated to a TA (*Transaction Agent*) module and all the TAs are grouped by knowledge areas and ruled by the MA (*Mediation Agent*) module. The system knowledge is based on a believe network (BN), formed by the particular knowledge of all these internal agents. Each time a request is made, based on the specific knowledge of the MA or/and the TA, the corresponding provider agents are located. The inverse case is also possible, when a provider makes an offer and the corresponding user agents are found. The following paragraph describes MAS architecture.

3. MAS (Multi-Agent System)

3.1 Introduction

Multi-agent systems are computational systems composed of several agents capable of mutual and environmental interactions. Agents can communicate, cooperate, coordinate and negotiate with one another, to reach both individual goals and the good of the overall system in which they are located.

In ABROSE, the MAS is concerned with the brokerage functionality for e-commerce activity. Buyers and sellers are represented by proactive, autonomous and cooperative agents. The originality of this broker is in the usage of a collective memory to find a relevant agent and in the learning process that updates the knowledge of the collective memory at several levels. Furthermore, each agent has its own viewpoint, of itself and of other agents in the system.

3.2 Architecture

It is a three-layer architecture, where every layer is made of communicating agents. At any level, an agent is composed of cooperative agents from a lower level (figure 3).

- *Mediation Agent* (MA), upper level: they control TAs, encapsulate all their acquired knowledge and will progressively learn from system transactions. The MA is also responsible for dispatching relevant messages to other MAs (or to a TA that it holds, supposing the MA believes that it can manage the message). The MA can be added or deleted from the system, depending on the needs of having a global knowledge (global BN).
- *Transaction Agent* (TA), medium level: each customer or provider is associated with a TA, storing knowledge about themselves and obtaining information from other TAs hosted by its own MA. The TA is responsible for autonomously exchanging requests or answers with other TAs. At TA level, the right organisation comes up when each TA is able to solve requests or to answer an offer received in messages, and always knows partners able to help it. It uses its own beliefs in order to find these partners.
- *Believe Agent* (BA), lower level: both MA and TAs, contain information about themselves and about other agents located in their own level. These beliefs describing the organization of the agent society, are called *Belief Networks* (BN) and are composed of BAs. The knowledge of an agent on other agents is continuously updated on the basis of the transaction flow. Thus, each layer improves its internal organisation in order to give efficiently and all-the-time relevant information.

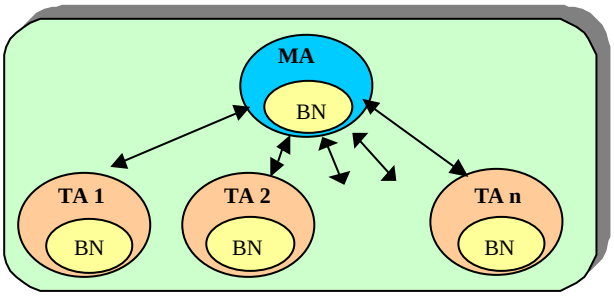

Figure 3. *MAS architecture.*

4. Communications in ABROSE

This section explains ABROSE communications design and implementation, introducing two new blocks UCOMMS and BCOMMS that compose the heart of this architecture, because they both are responsible for client and server being in touch (figure 4). We present user domain, broker domain and inter-domain communications architecture.

4.1 User Domain

In general terms, it consists of a web browser with a Java VM executing the different classes (modules CA, AMA,...) that is downloaded from the web server through HTTP. The first program to be called is an applet (thrown from ABROSE web access page). Afterwards, the rest of the modules are loaded in their correspondent graphical interface (CA or AMA) or in independent threads if no graphics are needed (UCOMMS).

In order to communicate between them, these modules use their Java object references, so the main point to allow communications is to provide a method to exchange these references. When the reference of the destination module is known, communication is as easy as executing the desired method in the elected module.

UCOMMS, responsible for communicating user and broker domain (see 4.3), communicates with the rest of the modules of the domain using these Java references as well.

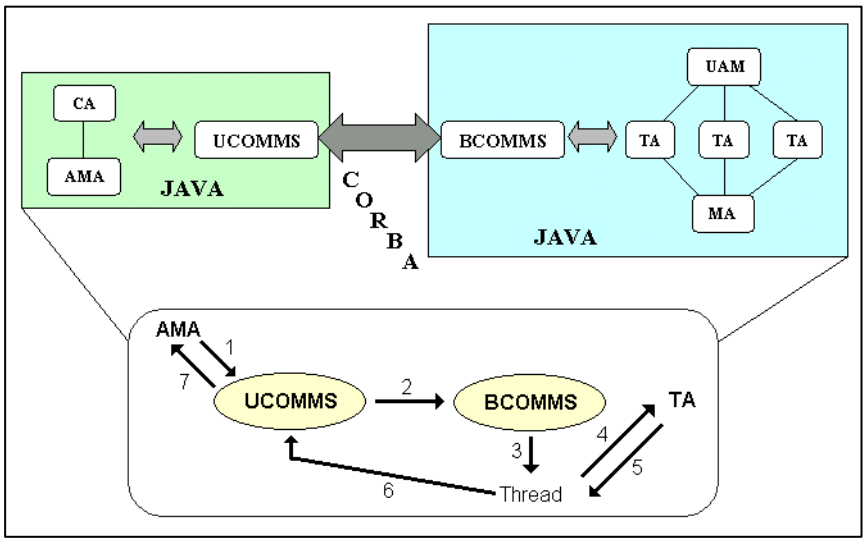

Figure 4. *Communications architecture and details.*

4.2 Broker Domain

Communications are done in the same terms as before. Every module is implemented as an independent thread (BM, BCOMMS, ...), although no graphical interface is used now.

The BM is the first launched component and it is responsible for initialising the rest of the modules and dispatching references between them. As mentioned, communications are done using these references to execute the corresponding methods.

BCOMMS is the symmetric module of UCOMMS in the broker domain. It is responsible for joining the user domain and the broker domain. Communications with the rest of the modules are done again, using Java references (including agent communications).

4.3 Client-server communications (inter-domain)

Communications between each domain have been implemented using CORBA and a centralized approach, with two modules (one for each domain) responsible for sending and receiving data from the other domain. That way, UCOMMS and BCOMMS are the only modules that need to speak CORBA.

As soon as users need to send any request to the broker, they invoke a method in UCOMMS. This module contacts BCOMMS, which delivers the query to the corresponding module inside broker domain.

4.4 Comparison

Another possibility would be to make every module responsible for its own communications [3]. Every module in the broker domain would be a CORBA object and communications between them would be through CORBA instead of invocations through Java references, so the system could take advantage of CORBA object distribution possibilities, enabling every module to run on a different machine. Every module in the client side would communicate at its own risk with the corresponding broker module.
CORBA utilization, however, implies a speed execution decrease compared to that we have in ABROSE scheme, where every broker module is executed in the same JVM and method invocations are immediately done.

With this distributed possibility, every CORBA module would have to define their programming interfaces using IDL and from the beginning of the implementation stage it would be stated which methods and parameters are accessible from other modules. This may imply a stronger effort in the design stage, but it makes system integration easier because it can be assured that at least module interconnections are homogeneously done.

With ABROSE scheme, the only components that must use CORBA are UCOMMS and BCOMMS, and so communications can be managed easier because the rest of the modules stay absolutely unaware of communications complexity (connections, sending and receiving data, error controls, etc.).

It must also be noted that ABROSE does not require a high distribution degree, so centralized advantages are more considered than distributed ones.

4.5 Multi-user

When the user gets into the system, a new UCOMMS instance is created. This module contacts BCOMMS passing its CORBA reference (and getting the equivalent reference from the broker) and from then on, both domains remain connected. From this interchange, it can be deduced that there will only be one BCOMMS instance executing at a time in the broker, while there will be one UCOMMS instance for every connected user. This increases considerably BCOMMS complexity, making it not only a bridge module devoted to receiving calls from UCOMMS and resending them to the proper module or receiving invocations from

the broker in order to send them back to the client. Since there can be several users simultaneously connected, BCOMMS must remain permanently unblocked and ready to receive requests from users or answer from the broker.

The applied solution (section 4.6 will present another view) goes through the creation of an independent Java thread every time BCOMMS receives a client call. This way, the complete process would be (figure 4):

- UCOMMS receives a request and transmits it to BCOMMS.
- Instead of making the request to the broker by itself (CORBA invocations can be made non-blocking, but Java invocations as these are, block the calling module until the end of the executed method), it creates a thread that is responsible for making the real request.
- This thread rests blocked, and BCOMMS is free again to keep on receiving more requests.

To send the answer back to the user it is not necessary to pass through BCOMMS again (so it will not suffer from an additional overflow). The auxiliary thread will receive the answer (it will usually be the attribute returned by the invoked Java method) and will directly give it back to UCOMMS.

4.6 Call-backs

In CORBA synchronous scheme, the answer to a request is usually the returned attribute of an invoked method (the client receives the answer as soon as the server method ends its execution). Occasionally however, it is very interesting to have this answer asynchronously and call-backs are the way to do it. If it is foreseeing that the request will take a long time before it can be processed, the client should not be kept waiting until the end of a specific method. It is much more efficient, from the user viewpoint, to get free once the request is made and make the server send the answer back by itself.

The problem in our case is that every Java call is, by definition, a blocking call. This means that the calling module cannot continue its execution until the module that incorporates the called method finishes its execution. In CORBA there is the possibility of defining a method as *one-way* that must obviously return nothing and doesn't block the caller (these are the methods normally used to implement call-backs).

Had we used CORBA to communicate every module (following the distributed approach presented in 4.4), no additional mechanism would have been needed (such as threads created by BCOMMS, in order not to get blocked).

5. Management

This section explains the existing problematic in agent-based application management like ABROSE. It also gives some approaches to its solution comparing several proposals that integrate network management with agent-based application management deployed over these networks, managing at time these applications with minimal additional effort of developers.

5.1 Problematical

From management viewpoint, the deployment of agent-based environments supposes the challenge of obtaining a behavior global view of the applications, through the management information maintained in its components. Therefore, it is necessary to standardize the access to that information, allowing the management service implementations being reused for future agent-based applications. This also implies the need to instrument the applications to provide the requested management information. On the other side, the big amount of existing management platforms requires these applications management to be integrated with traditional network management technologies. Another important requirement is to minimize the impact over applications so that management functionality affects their development as less as possible.

5.2 Possible solutions

The first approach developed in ABROSE (figure 5) has used a gateway that accessed the different components of the application through a single CORBA interface (the management dispatcher contained in the BM), forwarding the received requests to the application and to network and system agents.

In this case, there was a *matching* between component names and the defined management information so that the dispatcher could redirect requests using this straight relationship to the different components, threads in the same process address space. These application components might have a well-known management interface, so that access and modification to previously defined variables was possible.

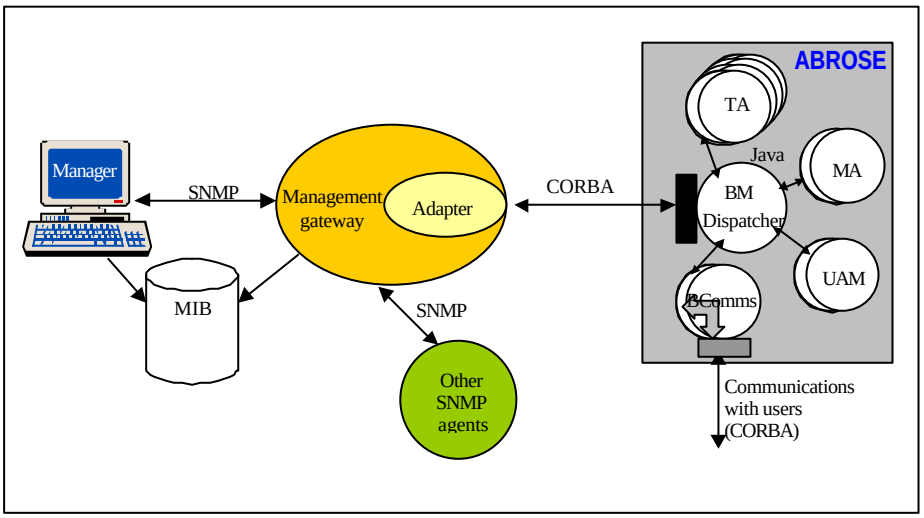

Figure 5. ***Adopted solution in the first version of ABROSE***

The instrumentation has been performed with the inheritance of objects that implement the management interfaces and maintain the managed variables [7].

For a highly distributed scenario (see section 4.4), Joint Inter-Domain Management (JIDM) [8] works should be considered: it defines how to achieve the interoperability between CORBA, SNMP and CMIP, describing the translation of the specifications and the interactions between these domains.

Applying JIDM concepts, the previous approach would work as follows. When the gateway receives a request, the JIDM algorithms defined for the interaction translation are used to access to the manageable resources, translating SNMP primitives to CORBA invocations to objects, obtaining their references through the services created for this purpose. Besides, the components of these resources would implement the IDL interfaces generated translating the specifications from the management information previously defined in a SMI MIB.

With this approach, CORBA interceptors can also be used to account the operations exchanged between different components. There exists the possibility of managing different instances of the same component, very useful in a multi-agent system. However, the rise of interactions between different system components will suppose a loss in the obtained performance [6].

5.3 Final approach

Another approach, which has been applied for the management of the final ABROSE prototype [9], is based on components technology. This facilitates software development, using previously developed objects with modifiable attributes accessing to interfaces that fulfill certain design patterns. In the particular case of Java, management beans have been defined in JMX (*Java Management Extensions*) for that purpose.

With this, it is possible to manage a multi-agent and multi-threaded application, transparently to managers and programmers, creating components based on the defined management information and finally, adding them to the different threads that compose the application. These components also have the ability of attending requests from traditional network management protocols such as SNMP. The existence of other components that forward requests to other applications allows the usage of external agents to obtain standard management information without requiring any gateway.

In relation to CORBA, the communications were monitored using interceptors. A thread was in charge of monitoring the users' connection, accessing the users' CORBA interface, so the manager could know in every moment who was connected, from where and how much time, being these data useful for a charging functionality.

Another thread was in charge of monitoring the states of each agent, asking it to them in a predefined period of time. This thread would send a notification to the manager if the monitored status was different from that contained in a table, so that he/she could know every moment if a certain agent was running, sleeping or stopped.

6. Conclusions

The main conclusions we can extract from communication experiences are related with the selection of a centralized communication scheme between domains and the utilization of Java inside of them.

The centralized scheme allows concentrating all the complexity that every system demanding connectivity has, just in two modules, so that all the others can remain unaware of communication architecture. As it usually happens in a centralized system, a little flexibility is lost and a little additional delay is introduced, because whenever a communication is to be established, these intermediate modules must be traversed.

The fact of using Java as the communication medium between modules inside each domain, allows higher speed than using CORBA but forces to replace call-backs versatility with a more complex implementation code. At the same time, application environment does not tend to be extremely distributed or heterogeneous, so that CORBA utilization in intra-domain communications could be justified.

Concerning to agent-based applications management, it has been shown that monitoring and controlling them results in an improvement in the quality of service demanded by its users. The existing technology allows performing this management by means of multiple approaches, being each one of them more adequate for certain situations.
We have also explained several approaches to manage agent-based distributed applications, through the utilization of management platforms and existing network management tools, using solutions based on gateways with different generalization level and components technology. The management instrumentation of them has been performed using concepts like inheritance and CORBA interceptors.

Both the presented communication and management architecture have already been conveniently validated and verified in the different prototypes implemented for ABROSE. The agent platform has also been implemented, tested and validated in ABROSE and results have partly been evaluated and reported to FIPA (*Foundation for Intelligent Physical Agent*) [10].

Acknowledgments

This work has been partially financed by the European Commission through ACTS ABROSE (AC316).

References

[1] *ABROSE: A Cooperative Multi-Agent Based Framework for E-Marketplace*. H.J. Einsiedler, A.Léger, M-P.Gleizes. Agents Technology in Europe. ACTS Activities. Infowin'99

[2] ACM "*Recommender Systems*". Special Issue of Communications of the Association of Computing Machinery – Vol40 N°3 – March 1997

[3] *ABS Broker Business Model*, D23. P.Alzon, P.Tothesan, M.Hubert, E.Athanassiou, A.H. Van Han. 1996 (http://b5www.berkom.de/ABS/D23.htm)

[4] *Application of TINA-C Computing and Service Architecture Concepts to the development of an Advanced Information Brokerage Service in the context of EC*. J.I.Asensio, J.I.Moreno, V.Villagrá, J.Redondo. A.Nolle, I.Tothezan, G.Karestos. Telecommunications Information Networking Architecture. Santiago de Chile, TINA'97 Conference.

[5] *An Approach to Electronic Brokerage in TINA Environments.* J.I.Asensio, V.Villagrá, J.I.Moreno, J.Berrocal. Fifth International Conference On Intelligence in Services and Networks. Lectured Notes in Computer Science 1430 (Springer). Antwerp,Belgium. IS&N'98

[6] *Experiences with SNMP-based integrated management of a CORBA-based electronic commerce application*. J.I.Asensio, V.Villagrá, J.E.López·de·Vergara, J.Berrocal. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, Boston-Massachusetts, U.S.A, IM'99.

[7] *Experiencias en la Gestión de Aplicaciones Distribuidas*. J.E.López·de·Vergara, V.Villagrá, J.I.Asensio, J.I.Moreno, J.Berrocal. Actas de IX Jornadas Telecom I+D, Barcelona, Spain, Telecom'99.

[8] *Inter-Domain Management: Specification Translation (JIDM_ST) and Interaction Translation (JIDM_IT)*. The Open Group. Open Group Technical Standard C802, January 2000.

[9] *Mecanismos de comunicación y gestión de servicio de un broker de información multi-agente.* F.Valera, J.I.Moreno, V.Villagrá, J.J.Berrocal. Libro de ponencias de II Jornadas de Ingeniería Telemática, ISBN:84-89315-14-0. Madrid. JITEL'99.

[10] *Brokerage communication in a cooperative multi-agent based mediation service: one example in ABROSE*. P.Glize, M-P.Gleizes, A.Léger. Foundation for Intelligent Physical Agent CFP6_016-1999.