# DYNAMIC SERVER SELECTION IN DISTRIBUTED OBJECT COMPUTING SYSTEMS

ELARBI BADIDI[1], RUDOLF. K. KELLER[1], PETER. G. KROPF[2], VINCENT V. DONGEN[3]

[1]*Départ. Info., Université de Montréal, Québec, Canada*
*{badidi,keller}@iro.umontreal.ca*

[2]*Départ. Info., Université Laval, Québec, Canada*
*kropf@ift.ulaval.ca*

[3]*Centre de Recherche Informatique de Montréal, Québec, Canada*
*vandonge@crim.ca*

**Abstract**
*In this paper, we address the problem of dynamic server selection among a set of object servers, as the ones defined by CORBA and DCOM, providing the same service type. These servers are not necessarily replicas but may have different interfaces and belong to different service providers. As a solution, we propose a novel architecture that we call LoDACE. This architecture has been designed to allow dynamic server selection and load sharing in distributed object computing environments. Specifically, the architecture prevents the occurrence of major load imbalances that can cause failures in distributed applications. The architecture is based on both the use of a trading service and the monitoring of the servers' load. Our interest in the trader is motivated by the need to discover object servers dynamically. LoDACE allows service requests to be processed by lightly loaded servers selected dynamically. The expected results are better performances in terms of responsiveness and availability of servers.*

***Key words****: server selection, load sharing, distributed object computing.*

## 1. INTRODUCTION

Popular distributed services often suffer degraded performance in terms of weak responsiveness and reduced availability of servers. This problem is essentially obvious in the Internet where popular servers and network links are often saturated with the increasing service demands. Many of today's industrial applications require a high degree of availability. Server crashes and network failures are serious problems that frequently occur in distributed systems and for which great attention should be given to avoid their occurrence.
With the advent of Distributed Object Computing technologies (DOC), such as CORBA and DCOM, object servers are deployed to serve various clients written in different languages [11][14]. For example, a client program written in C++ and a client program written in Java, as an applet, or in Smalltalk, may invoke the services provided by the same object server. This

feature of DOC systems increases the flexibility for deploying object servers to be used by a big population of clients. Therefore, distributed systems have to scale in terms of both the number of services to be deployed and the number of users. In this context, availability and performance are significant issues.

Caching and replication are two techniques often used to manage services. Replication is a frequently suggested solution to problems of scale associated with distributed services. Replicas are created to improve performance and availability of servers. If a server is down and a replica of that server is available on another host of the system, the client request can still be satisfied. When a service is replicated, clients face the additional task of finding the best provider of that service. Many properties may be associated to a service such as its cost, the rate at which it is given, and the quality of the service. For a client, an optimal server is a server providing the required service with the minimal cost, with the minimal latency (the elapsed time between service request and service response), and with the best quality of service. Finding this optimal server is not an easy task and closely depends on the distributed system used. For clients response time is often the most predominant factor that has to be considered.

In this paper, we focus on a particular aspect of the problem: given a service type, how can we locate servers providing this service type, and how can we distribute service requests in a way that minimizes the average response time for client requests and increases availability of these servers?

Static server selection may be effective in small environments where the number of servers is small. However, in large distributed systems such as the Internet, servers may be added or removed dynamically. This increases the complexity of the problem. Server location has to be dynamic; therefore, server selection should be carried out dynamically. In the following, we will focus on this aspect of dynamic server selection in distributed object computing systems.

In DOC environments, distributed applications consist of objects (or components in the case of applications based on DCOM, JavaBeans, and alike) running across various clients and servers. Servers provide objects to be used by clients and by other servers. When one object calls an operation on another object, the former is frequently referred to as the client object, and the latter as the server (or target) object. Objects may be implemented in different programming languages, such as C++, Java, Smalltalk, and Ada. Once a request is issued by a client, an object server should be chosen from a set of object servers that are able to handle the client request.

The remainder of this paper is organized as follows: in section 2, we describe two mechanisms for locating objects in distributed systems. In section 3, we present four techniques for the selection of an object among a set of objects. In section 4, we detail our architecture for load-based dynamic selection of objects in a DOC environment. A prototype of the architecture is presented in section 5. In section 6, we review some related work, and finally, in section 7, we conclude the paper by discussing future work.

## 2. SERVER LOCATION TECHNIQUES

Two principal services standardized by ISO and by OMG, the naming service (or directory service) and the trading service, allow locating objects in a distributed object system.

The *naming service* is a mechanism which makes it possible for the objects of a distributed

application to locate other objects (local or remote) by means of their names [12]. The names are humanly recognizable values associated with an object. An object is identified by means of a reference object (i.e. an internal structure of identification that contains a fixed set of fields). The resolution of a name consists in determining the reference object associated with this name.

The *trading service* is a service that allows clients (or users) to dynamically discover offered services. The object providing this service this service is called a trader. A *trader* is a mechanism that facilitates the advertisement and the discovery of services. It communicates with servers (or service providers), with customers (users of services), and with other traders [9][10][13][4]. When a server wishes to announce its service, it records its offer in the trader's database. An offer of service contains a type of service, an identifier of the interface of the service where the service is provided, and the values of the service properties. The advertisement of a service is called *exportation*. When a customer requires a service, it sends a request of service to the trader to find a suitable service. A request of service expresses the characteristics required by the customer by specifying the type of the desired service and the constraints on the service properties. This operation is called *importation*. The trader, then, searches in its database service offers that match the customer's request. The list of the found service offers is then returned to the customer, which can choose from this list a server to which its service requests are to be sent.

## 3. SERVER SELECTION TECHNIQUES

A simple strategy for dynamic server selection is *random selection* among a set of replicas. It is being used, for example, by many FTP sites to distribute the load; giving to the user a list of sites where he/she can download files and let him/her select sites at random. It has been proven to be a practical fashion to distribute the load. In our context, once servers, providing a certain service type, have been located using the trading service, this policy may be used by clients to select a server from the list of servers found by the trading service.

A second strategy is *round-robin selection* among a set of replicas. In this policy, servers are selected in a cyclical fashion. Orbix OTM, for example, uses round-robin selection to implement load balancing on both an intra-host and a cross host basis [8]. DNS round-robin policy is used by web servers for load sharing purposes. Its purpose is to allow use of multiple HTTP servers, with identical contents, in order to distribute the connection loads. This is a simple strategy, which is expected to yield significant performance enhancements over the no-load sharing case and a slightly better distribution over the random selection policy.

A third strategy is the *bidding selection policy* in which a client or an intermediary agent, which acts on behalf of the client, starts an auction and broadcasts a request for bids which includes a description of the job to be run and an estimate of the required processing time. Servers reply with bids containing estimated completion times for the client job. After a pre-determined amount of time, the auction is closed and bids are evaluated. The auction's winner is selected for processing the client request. Enterprise is a decentralized scheduler for load-sharing in distributed computing environments which uses this strategy [15].

To have a better distribution of the load, we can add some intelligence to the server selection process by taking into account the load of servers. This strategy assumes that there is a means to know an approximate load value of each server providing a required service. Based on this

knowledge, service requests are distributed to least loaded servers. Load sharing between servers allows to improve the performance in distributed applications and to increase the availability of servers. Work on load distribution issues during the last two decades shows the benefits of load sharing over the no load sharing policy. In the following, we describe a system, that we call *LoDACE*, which allows dynamic server selection and load sharing among a set of object servers providing the same service type. This system uses the trading service for dynamic server location.

## 4. A LOAD-BASED DYNAMIC SERVER SELECTION ARCHITECTURE

To deal with the issue of load-based dynamic server selection in the context of DOC environments, we present in this section the design of a load sharing architecture that we call: the *LoDACE* architecture. LoDACE stands for ***Lo***ad ***D***istribution ***A***rchitecture for a Distributed ***C***omputing ***E***nvironment. We have based this architecture on the use of the trading service. The trader's service offers repository contains a full description of the services provided by different servers. The trader allows servers to register their service offers and enables clients to get services without requiring prior knowledge of both the location and the availability of any particular server. The aim of this architecture is to be as general as possible without being specific to a particular service type. In a distributed system, traders are organized in federations of traders [9], each of them managing a specific domain. For the time being, dynamic server selection and load sharing in our architecture are carried out for the servers within the domain of one single trader. Scalability issues are on our agenda of future work.

Real-time knowledge of each server load is essential in selecting lightly loaded servers to process service requests. Load information include, for example, the size of the input queue of the server, the input/output activity, the available real memory, the CPU load, and other parameters. Load information is managed by a load manager component. Fig. 1 presents a simplified architecture for dynamic server selection.

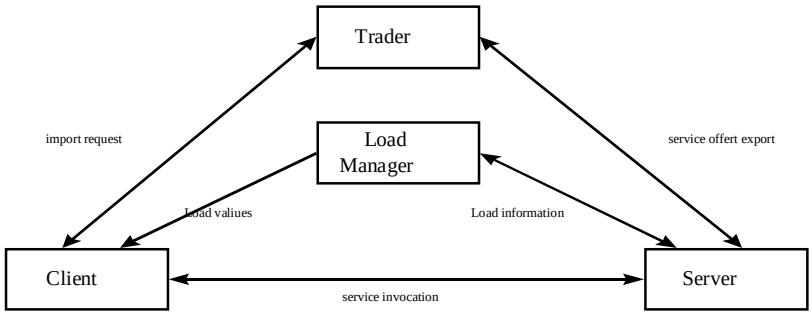


**Figure 1- Simplified architecture for dynamic server selection**

The *load manager* is the component responsible for gathering dynamic load information of servers, and providing clients (or other entities of the system) with information regarding load of servers. Load information is collected by polling servers or it is sent periodically by servers to the load manager. In this scheme, load distribution is not transparent to the client since it has to get load information of the servers responding to its import request from the load manager. The least loaded server is then selected to process service requests. In addition, each server has

to implement a load monitor to evaluate its load. Transparency can be achieved by introducing an additional component, that we call a *binder,* acting on behalf of the client. Likewise, to avoid loading the server by the exchange of periodic information with the load manager, we introduce an additional component, that we call a *load monitor*, common to all servers residing in the same host. Its function is to evaluate the load of each server.

## 4.1 LoDACE Architecture

Fig. 2 illustrates our LoDACE basic architecture which consists of the following components: a Trader, a Binder, a Load Manager (LMG), Load Monitors (LM), and Host Load Monitors (HLM). Requests and load information are exchanged between these components through an Object Request Broker (ORB).

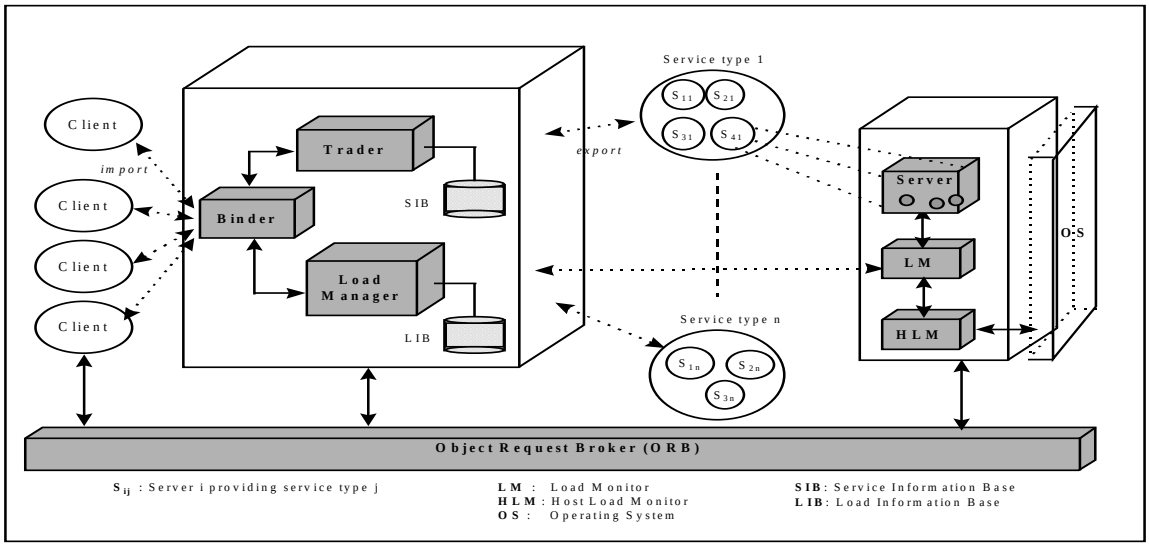


**Figure 2- LoDACE architecture**

The *binder* represents the interface between importers and the LoDACE system. It allows a client to transparently access and invoke services; that is, it provides the client with the import functionality of a trader. An importer only needs to interact with the binder to satisfy all its required operations.

A *load monitor* should be present in every host of the system. Its role is to determine the dynamic load of the host and all its servers. This load information is periodically transferred to the load manager. Similarly, a *host load monitor* should be present in every host of the system. It is used to compute the host's load value from the operating system's statistics. This value is sent to the load monitor within the same host. Load information is exchanged between load monitors and the load manager through the ORB because of the simplicity of communication, the heterogeneity of hardware and operating systems, and its independence of low level communication details.

## 4.2 Dynamic server selection

A client requiring a certain service type sends its request to the LoDACE system in order to

get the reference object of a server providing its required service. The client may be a user using a graphical user interface such as a Java applet, a classical process, or an object of a distributed application. The client request is sent to the binder which forwards it to the trader *lookup* interface. The trader returns back to the binder the service offers responding to the required service according to the constraints, preferences and policies specified by the client. Each service offer describes the service properties and the reference object of the server providing the service.
The binder, after that, invokes the load manager in order to get the reference object of the lightly loaded server at that moment. The load manager maintains a load information base (LIB) for all registered servers. This LIB serves to determine the lightly loaded server whose reference object is returned to the binder and then to the client. From this point, the client can initiate the connection to the server to get its required service. In this way, requests are processed by lightly loaded servers.

## 5. VALIDATION OF CONCEPTS

In order to evaluate our proposed architecture, an experimental prototype is under construction. The programming environment is OrbixWeb [6] and OrbixTrader [7] from Iona Technologies. OrbixWeb is an implementation of the OMG Common Object Request Broker Architecture (CORBA) that maps CORBA functionality to the Java programming language. OrbixTrader is an Iona implementation of the CORBA Trading Object Service. Fig. 3 illustrates the exchanged messages between the elements of this prototype in order to achieve dynamic server selection for client requests; and, therefore, load sharing between object servers providing the same service type.
The IDL specification describes the attributes and the methods of each component of the LoDACE system. Methods represent services provided by a component. The compilation of the IDL specification of each component generates stubs and skeletons for that component in the Java language.

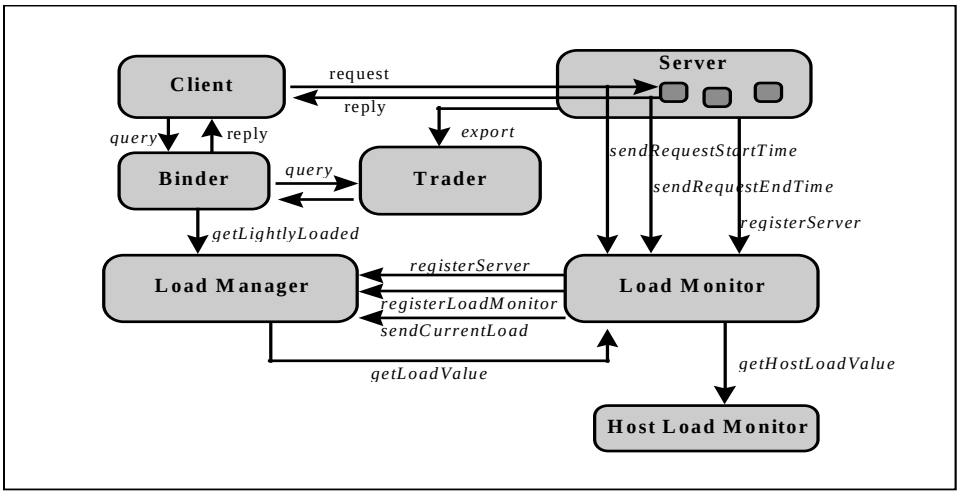


**Figure 3- LoDACE prototype**

Each load monitor (LM) of the system maintains a table of the servers which reside in the same host as the LM and which have exported their service offers to the trader. A unique number

identifies each server. Several servers providing the same service type may coexist in the same machine. Monitoring the load of each server is implemented using the OrbixWeb filter mechanism. Each time a request is received by any object of a server, the LM is notified of the arrival of a new request to the server by means of the *sendRequestStartTime()* LM operation. After processing the request, the server notifies the LM by invoking the *sendRequestEnd-Time()* LM operation. These two operations allow computing the busy time of a server during a specific period.

The load manager maintains three tables: the load monitors' table, the servers' table, and the load information table. The first table identifies all the load monitors that are known by the load manager. The second one identifies all the known servers with the associated load monitors. The third one keeps the last load value, calculated by load monitors, for each registered server. The load manager has an operation called *getLightlyLoaded()* that can be invoked in order to determine the lightly loaded server from a sequence of servers provided as input parameters of this operation.

The binder implements an interface similar to the trader *lookup* interface. However, the query method, inherited from this interface, which allows querying service offers registered in the trader, has been extended. The binder communicates with the load manager to get the service offer associated to the lightly loaded server among the service offers returned by the trader. Fig. 4 shows an applet for the binder interface. The first part of this applet offers a means to select a service type from the list of service types maintained by the trader in its service type repository. The second part allows the client to specify its constraints, i.e. an expression over the service type properties, and the client preferences, i.e. for example how to order the resulting offers. A request is then built and sent to the trader.

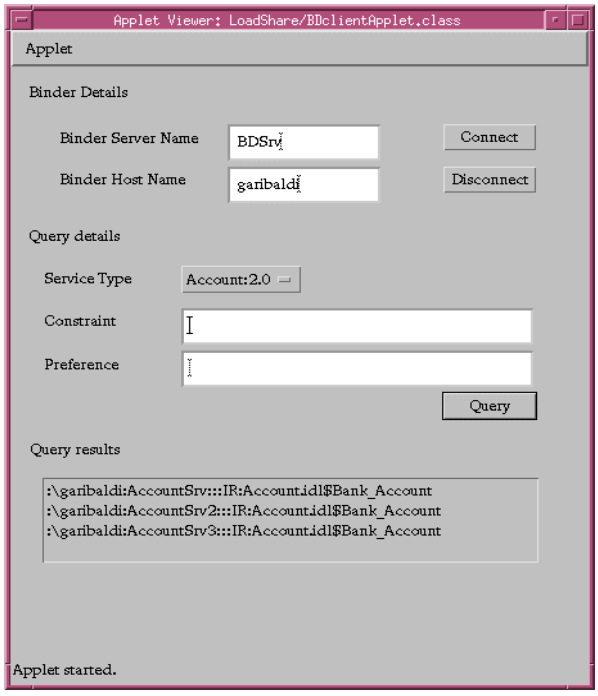


**Figure 4 - Binder Applet**

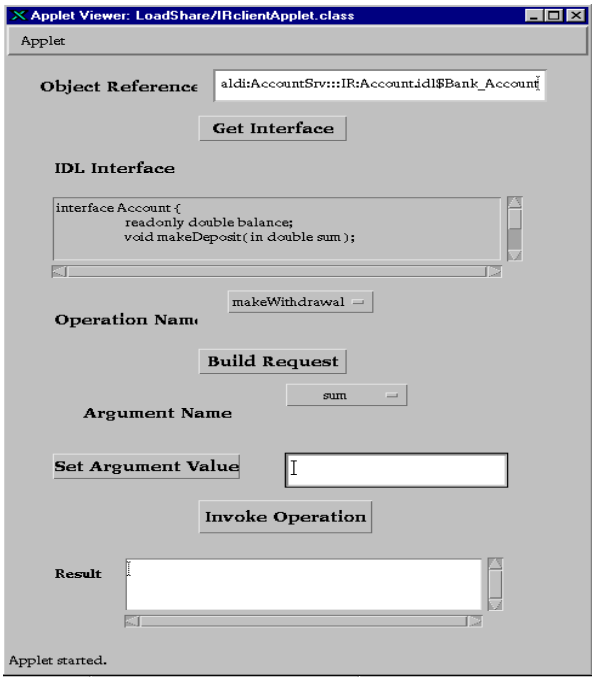


**Figure 5- Dynamic service invocation**

The trader list of servers providing the required service is displayed and ordered according to the load of each server. The first element of this list is the least loaded server and so on. Using

the dynamic invocation interface, the client can discover the interface provided by the selected service and build service requests to get the required service. Fig. 5 shows the interface used by clients to build service requests.

## 6. RELATED WORK

Throughout the last two decades, a number of projects considered the problems of locating objects and distributing load in locally distributed environments. The NetSolve system at the University of Tennessee is a client-server architecture designed to allow clients in a distributed system to execute scientific problems remotely [5]. A broker with respect to load information of servers carries out computational server selection. NetSolve shares with LoDACE similar goals in terms of service mediation and load based server selection. Unlike NetSolve, LoDACE prototype is implemented in a CORBA environment. All communications, of requests and data, are made via the ORB; and clients and servers may be implemented using different programming languages. Communications in the NetSolve system are made through the UNIX socket layer and the XDR protocol.

The SuperWeb system at the University of California in Santa Barbara is a project whose goal is to build a global computing infrastructure over the Internet [1]. SuperWeb aims to allow machines connected to the Internet to make a portion of their resources available for remote clients, and to use dedicated resources from other machines. Mediation of resources and server selection are performed by a broker. Load based selection is not considered in SuperWeb.

The LYDIA project [2][3] describes how load balancing can be integrated in IDL environments such as CORBA. The approach considered in LYDIA is to bring some modifications to the source code of the stubs and the skeletons generated by the IDL compiler in a way that servers and clients be instrumented to measure their dynamic load. Load information is reported to a load balancer that cooperates with the Naming Service for the selection of an appropriate object reference out of a number of alternatives. In LoDACE, we think that since there are different CORBA implementations, load monitoring should be made by other mechanisms rather than by modifying stubs and skeletons. These stubs and skeletons are not standardized and are generated for a target programming language. Monitors in our prototype are CORBA objects with defined interfaces, and servers instrumentation is made using the Orbix filter mechanism. In addition, the trading service is more convenient for server discovery on the base of service descriptions and clients constraints.

## 7. CONCLUSION

This paper has described the issue of dynamic server selection in a DOC environment where object servers are running on heterogeneous machines and may provide the same service type. We have proposed an architecture, that we call LoDACE, to provide load-based dynamic server selection in such environments. This architecture is based on the use of both the trading service and the monitoring of the servers load. We have also presented a prototype under construction. The experimental platform is an OrbixWeb environment from Iona. OrbixWeb is a CORBA2.0 compliant ORB that maps CORBA functionality to the Java programming language.

At the time of the writing of this paper, we have incepted experimentation work to evaluate the

performance of our architecture. We also intend to extend LoDACE to support load sharing techniques, such as server initiated and receiver initiated policies, and to consider fault tolerance and scalability issues.

## REFERENCES


[1] A.D. Alexandrov, M. Ibel, K.E. Schauser, and C.J. Scheiman: "*SuperWeb: Research Issues in Java-Based Global Computing*".- In: Concurrency: Practice and Experience, Vol 9, Issue 6, pp 535-553, (1997).

[2] B. Schiemann: "*Requirement for an Interface Definition Language Environment for Load Balancing (Second Draft)*".- ESPRIT III P8144, LYDIA/WP.4/T.4.1/D5, June (1996).

[3] B. Schiemann: "*Specification of IDL Mechanisms for Load Balancing (First Draft)*".- ESPRIT III P8144, LYDIA/WP.4/T.4.2/D6, July (1996).

[4] G. Outhred and J. Potter: "*An entreprise trader model for DCOM*".- In: Proc. Of the IFIP/IEEE Intern. Conference on Open Distributed Processing and Distributed Platforms, Toronto, Canada. pp 63-73, May (1997).

[5] H. Casanova and J. Dongarra: "*NetSolve: A Network Server for Solving Computational Science Problems*".- In: the International Journal of Supercomputer Applications and High Performance Computing, Volume 11, Number 3, pp 212-223, (1997).

[6] Iona Technologies Ltd.: "*OrbixWeb programming guide*".- November, (1996).

[7] Iona Technologies PLC: "*OrbixTrader Programmer's Guide and Reference*".- August (1997).

[8] Iona Technologies: "*The Orbix Object Transaction Monitor (OTM)*".- White paper, version 1.0, April (1997).

[9] ITU/ISO: "*ODP Trading Function Part 1: Specification*".- ISO/IEC 2nd DIS 13235-1, ITU-T Draft Rec. X950-1, (1996).

[10] ITU/ISO: "*Reference Model of Open Distributed Processing Part 1: Overview*".- ISO/IEC 10746-1, ITU-T Rec. X901, (1996).

[11] Microsoft Corporation: "*Microsoft Windows NT Server, DCOM Technical Overview*".- White Paper, (1996).

[12] OMG: "*CORBAservices: Common Object Services Specification*".- Updated version: July (1997).

[13] OMG: "*RFP5 Submission Trading Object Service*".- OMG Document orbos/960506, OMG, Framingham, MA, (1996).

[14] OMG: "*The Common Object Request Broker: Architecture and Specification*".- Rev.2.1, August (1997).

[15] T. W. Malone, R. E. Fikes and M. T. Howard: "*Enterprise: A market-like task scheduler for distributed computing environments*".- In: B. A. Huberman, editor, The Ecology of Computation, pp 177-205. North-Holland, Amsterdam, (1988).