QoS and Reliability Models for Network Computing

Ladislau Bölöni, Kyung Koo Jun, and Dan C. Marinescu
(boloni, junkk, dcm@cs.purdue.edu
Computer Sciences Department
Purdue University, West Lafayette, IN, 47907, USA

Abstract

We present a taxonomy of the agents involved in network computing, introduce models capable to describe different levels of reliability and Quality of Service required by a service contract and discuss architectural requirements for each model.

Background

The term network computing is rather vague, it implies a collection of heterogeneous computing and communication systems capable to provide a wide variety of services to a community of users with diverse needs and abilities, subject to a set of constrains as, reliability, security, guarantees of quality of service, ease of use. The very idea that network computing is feasible can be traced to the success of the World Wide Web which uses the Internet, a proven networking technology, to provide access to multimedia information, text, images video, and audio distributed world wide.

The Web required the development of browsers, the user access software able to run on most platforms, of a new communication protocol, HTTP[8], and of Web servers. The palette of technologies necessary for network computing is considerably richer. Heterogeneity of hardware and software systems poses insurmountable problems to the designer of a distributed system. Only recently the development of Java platforms running under virtually any operating system and machine architecture allowed code mobility. Java extensions including the JavaServer Architecture provides now the necessary framework for designing servers with a required functionality. Inter-agent operability supported by CORBA and IIOP[9], the Internet Inter Orb Protocol, and of environments which integrate CORBA and Java, like the Visigenic's VisiBroker [10] or Sun Microsystems JavaIDL [11], are necessary milestones that bring network computing within reach. Yet all these enabling technologies need to be complemented by studies of the computational models to be supported by a network computing system. From these models we need to draw conclusions about the architecture of network computing systems.

The scope of network computing is considerably more complex than that of the Web, it is not limited to moving data, it needs to move both data computations around the network and ensure interoperability among services created by different organizations and individuals. Instead of providing a limited number of services with no reliability and quality of service guarantees, network computing implies open ended systems able to integrate new services as they become available, capable to provide reliable services and guarantee the quality of service if so requested by an application. One expects a network computing system to accommodate a wide range of service options for the quality of service and reliability, the same way computer network support transport protocols with different service guarantees e.g. datagrams and virtual circuits. Some of the services built into the system are internal services needed to ensure integrity, security, and optimal use of resources. Performing such functions require detailed knowledge about the users of the system, the hardware platforms and communication channels, as well as the software available.

To understand the interactions between a user and a network computer we need to develop models of the activities required by a user and draw conclusions concerning the architecture of the hardware and software systems capable to satisfy such requests.

Contracts and Agents

Extensions of the ubiquitous client server paradigm suitable for network computing are discussed below. We use the following terminology and notations. *A contract is* an activity involving one or more agents making requests, *client(s),* and one or more agents capable to satisfy the requests and provide *services*, called *server(s).*

In the original client server paradigm the most common form of interactions are pairwise interactions, a client interacts with one server at a time. The *lifetime* of the agents involved is longer than the duration of the contract, the agents are assumed to be *reliable* in the sense that once a contract has been accepted both parties are expected to be alive until its completion. The agents involved in a contract are *immobile*. Current distributed systems are based upon *stateless servers* and use transactions that either complete or fail to complete. Transactions are often implemented as Remote Procedure Calls, supported by RPC protocols and the client has the responsibility to maintain state and reissue a transaction that failed to complete. Operating systems like UNIX or Windows NT support a set of common services on each host at known ports e.g. an HTTP server listens to port 80, ftp to port 21, telnet to port 23, mail to port 25 and so on. There are also specialized servers e.g. ftp servers, database servers running on dedicated hosts. The address of such a server can be found with the aid of a *name server.* The burden of coordinating a complex contract in such an environment falls on the user of the system.

.

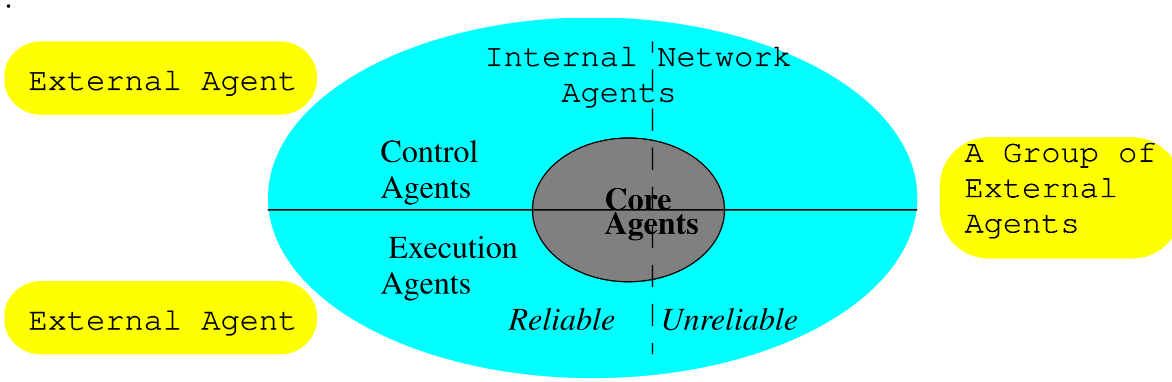

Figure 1. A taxonomy of agents involved in network computing.

The simplest form of a contract is a *transaction* involving one client and one server. More complex contracts may be expressed as *activity flow graphs* describing a set of atomic activities and may involve an external agent and a network of servers or possibly a group of external agents and a network of servers as in the case of collaborative environments. Most dogmas of the traditional client server model are no longer true in network computing. A client may issue a number of contracts at the same time. Each contract may be complex and require that each server becomes in turn a client to a second stage server which in turn becomes a client for the next stage server and so on. In multiple stage systems all servers involved in a contract form a network and need to maintain some state information. Some of the agents provided by a network computing system may be unreliable. A taxonomy of the agents involved in network computing is presented in Figure 1 and the agent types are defined below.

A network computing system is expected to provide a set of *basic services* and allow the development of *Intranets* capable of providing *augmented services*. The development of *Intranet services* is discussed below. Some of the services are *permanent.* Other services are *transient*, they are started at the request of a given application.

In network computing we distinguish between *external agents*, entities existing outside the system and providing the interface with the users and *internal network agents* capable to provide services. A desirable property of a network computing system is to support *light weight external agents*, in other words to allow the developer of an Intranet application to create complex functions without the need to write complex programs. To accommodate contracts requiring a dependable system we need *control agents* which are not directly involved in a contract but supervise the entire system. *Execution agents* are directly implicated in a contract. Once a contract is accepted, the control agents ensure the that the contract is properly mapped into services offered by the Intranet, and that the QoS and reliability requirements of the contract are satisfied.

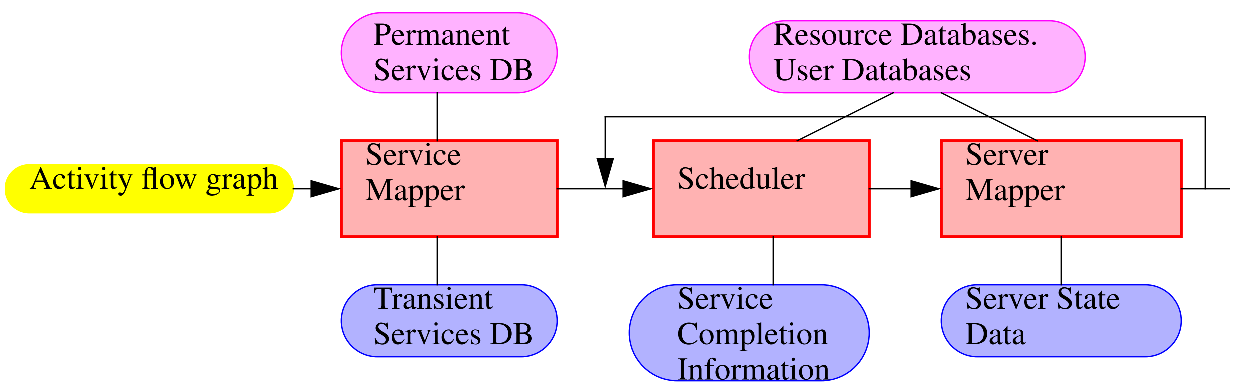

Figure 2. Processing of a complex contract described by means of an activity flow graph. The service mapper uses databases of permanent and transient services to map the atomic activities onto services. Then the scheduler creates a scheduling object. The scheduler and the server mapper use static and dynamic information to carry out services. The static information is provided by resource and user databases. The dynamic information relative to the state of the servers and the completion of atomic activities is collected by QoS monitors.

Service mapping establishes a correspondence between atomic activities required by a contract and services available in a given system. *Scheduling* is the process of deciding *when* to perform each atomic activity. An atomic activity is *schedulable/ready* if all its data dependencies are satisfied. Several *scheduling policies* are possible: *greedy scheduling*, schedule all ready activities at the earliest time, *optimal scheduling,* attempt to optimize a cost function. The objective may be to minimize the total execution time, to maximize the resource utilization of the system, or to schedule a set of concurrent activities which have a common consumer of their results such that all will complete at the same time. *Server mapping* is the process of deciding *where* to perform each atomic activity. Service mapping, scheduling and server mapping are closely related and interdependent, and they can be done by a single agent; we'll call this aggregate activity scheduling.

The aggregate activity scheduling can be static or dynamic. In the case of *static scheduling* we assume that we have all the required information to make all the mapping and scheduling decisions before the beginning of the execution of the contract. The *dynamic scheduling* reflects the ability of the agent to change the scheduling and mapping decisions during the lifetime of a contract.

Scheduling a contract can be done by an external agent by establishing point to point connections to individual servers and coordinate the entire process from the client site. This approach is likely to be highly inefficient, require a heavy involvement of the user, and leads to *heavyweight external agents*. Figure 3 presents such a system where there is no cooperation among network agents. Each server registers its services with a name server, then the external agent locates services using the name server. A *light weight external agent* is one which needs to maintain relatively simple state regardless of the QoS and reliability requirements of the contract. This can only be supported if enough control services are provided by the Intranet. Figure 4 illustrates such a contract. The Dispatching service maintains a virtual presence of the temporary external agent which may leave after submitting the contract. The Scheduling service maps the contract and schedules atomic activities. Services at different stages of a contract cooperate with one another. The System Monitoring service ensures fault tolerance.

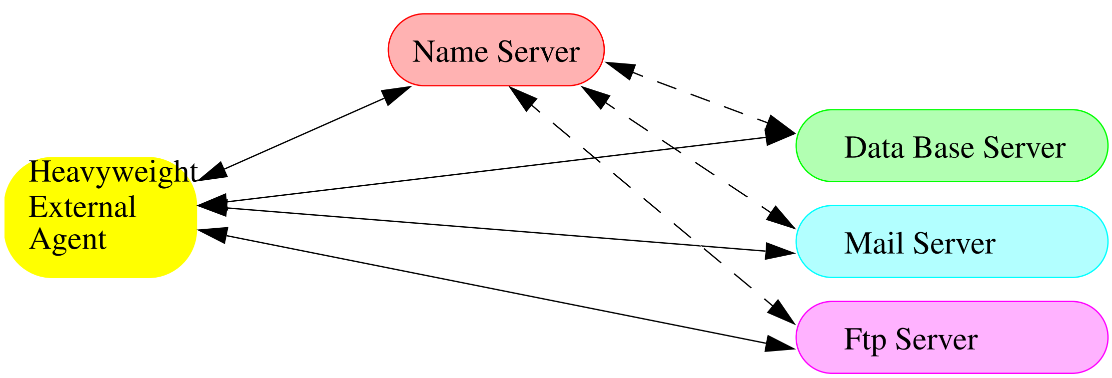

Figure 3. A heavyweight external agent coordinates the entire execution of the contract.The network does not provide any guarantees for the quality of service or reliability.

A contract may require firm Quality of Service, QoS commitments, a best effort approach, or no QoS. To ensure a firm QoS requirements we need a special class of control agents called *brokers* capable to negotiate with system management agents the reservation of the critical resources needed for the contract. *QoS monitors* are needed to supervise the execution and coordinate agent migration in case of best effort requirements.

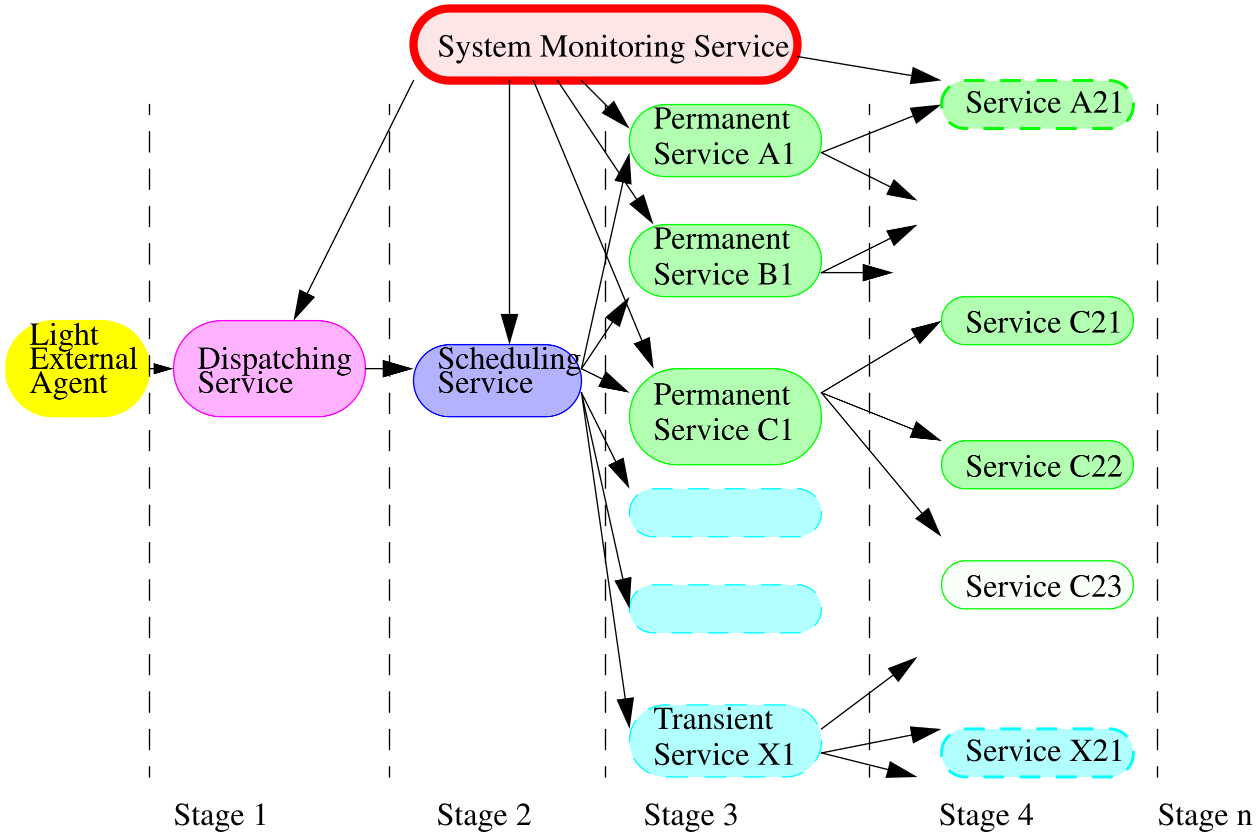

Figure 4. A lightweight external agent requests a reliable contract from an Intranet with control agent support.

An intelligent agent is one capable to process knowledge. Network computing is inconceivable without some level of intelligence. Agent interoperability, scheduling, load balancing, fault tolerance, and other control functions are difficult if not impossible to perform without access to knowledge bases. An intelligent network computing system is one capable to process complex contracts, provide a set of network services necessary to support light weight external agents, allow agents to cooperate with one another, and ensure Quality of Service, QoS and fault tolerance.

We assume that multiple network servers are involved in each contract and multi-stage interactions among agents are supported. We characterize each contract by a string describing, the number of external agents implicated, the desired reliability and QoS. The first symbol denotes the number of external agents, there are Single (S) and Multiple external agents (M) contracts. The second symbol reflects if the contract requires reliable (R) services or it is content with Unreliable (U) service. The third symbol reflects whether the contract requires guarantees of the Quality of service (Q), is content with a Best effort (B) or requires No guarantees (N). An intelligent network computing system should be capable to support a wide range of contracts from SNN to SRQ and MRQ contracts. A metacomputing environment is likely to be based upon SRQ contracts, network commerce on MRB contracts and a collaborative environment on MRQ contracts.

Classifications based upon different criteria are possible yet we believe that the three criteria we choose are paramount for the characterization of each model. Supporting contracts with multiple external agents has major implications upon the design of a system. An *arbitrator* capable to deal with possibly conflicting requests from several external agents must be embedded into such a system. If the contract requires reliable services we need mechanisms to ensure the virtual presence of the corresponding agents. Once the external agent is unreliable there is the need that another agent acts as a proxy and maintains the state of the contract. Once services are performed by unreliable agents there is the need for some *system monitor* capable to keep track of the overall system configuration. Quality of service requirements can only be satisfied if there are abundant system resources and if reservation schemes can be enforced by *brokers* and *QoS monitors*.

Servers and Services

A network computing system is expected to provide several layers of services, see Figure 5. The *basic network services* provide the interface to the communication and computing hardware and are essential for the functionality and integrity of the system. Transport functions as well as *ab inito servers* running on each system, able to report the properties of its host, and start processes on that host belong to this layer.

.

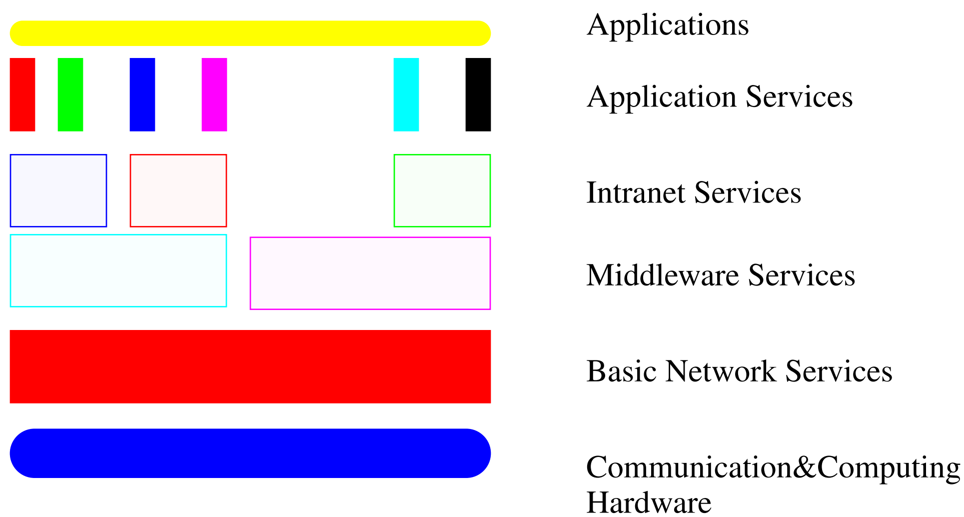

Figure 5. The hierarchy of services.

Middleware services, MS, form the next layer. They are expected to provide primitives for creating control agents and allow control agents in different Intranets to collaborate with each other. Examples of middleware services are those provided by the Infospheres system developed at Caltech by Mani Chandy's group [], or the JavaLite package [].

Intranet services, IS, are performed by control and permanent execution agents specific to a particular class of applications. The control agents support reliability and QoS requirements. *Application services, AS,* consists primarily of temporary execution agents started upon request when a contract is accepted.

A brief discussion of each of the Intranet services presented in Table 1 follows. We start by discussing the System Monitor. To ensure that core services are resilient to physical server failure the system should rely on a system monitor capable to (a) detect when a server providing a core service has failed, (b) identify a capable server and restart the service at that particular network address, and (c) notify all agents how to obtain services in the updated configuration. The System Monitor can provide *active monitoring* by polling periodically all servers providing core services or *passive monitoring*, namely expecting periodic messages from all servers in its *configuration file* and timing out if such a message is not received. The Monitor needs also a *network resource file* to identify capable platforms for each core service and the mechanism to restart each service.

The Monitor itself provides a critical service and needs to be replicated. A possible strategy is to have one or more *stand by Monitors* sharing the core configuration and the network resource files with the active monitor. The stand by Monitors run an election algorithm whenever they detect the failure of the active Monitor.

The Scheduler is responsible for what we have defined as aggregate activity scheduling. There may be one or more scheduling agents. In the case of more than one scheduling agent an alternative is to have distributed scheduling, the scheduling decisions are made locally for each agent. Another possibility is to is divide the application into clusters, each with his own scheduler. We call this multilevel scheduling. In fact we always have at least two levels of scheduling because the agents are scheduled locally by the scheduler of the system where they execute. If there are QoS requirements the scheduler agents communicate with the system monitor and the broker agent. The scheduling strategy implemented should permit the dynamic adaptation of the schedule in function of the state of the system.

A Broker tries to optimize the execution at the agent level by finding a host for the execution and negotiating with the local operation system the reservation of the critical resources. In some sense, the broker can take over some of the mapping tasks of the scheduler. The broker agent is always needed for soft or hard real time systems.

The QoS Monitors are monitoring the progress of the execution of an agent. Intelligent agents can usually perform self-monitoring. For legacy applications, some (usually very limited) monitoring information can be obtained from the local operating system. If a more accurate supervision is needed, a dedicated agent (wrapper) is needed.

The QoS Monitors collect performance data from different agents implicated in a contract. They interact with the scheduler and as a result several actions are possible: additional resource reservations are made and the agent migrates to a different host, the scheduler produces a new schedule, the external agent is informed that the QoS requirements cannot be met.

In our model the initiators of all contracts are external agents, usually associated with the human users. However their requirements - which usually will be translated into scheduling decisions, may be conflicting. The role of the Arbitrator agents is to balance the requests according to the relative priority of different external agents and the current status of the system.

The dispatcher is the component capable to represent all external agents connected to it, in all contracts after the external agents disappear. The dispatcher should maintain private user information, e.g. all the information necessary to start a transient agent executing in behalf of a user. It should be able to make decisions in behalf of a user, e.g. decide if actions should be taken, e.g. initiate the actual buying of an item or stock when the conditions set by the user are fulfilled. When the contract cannot complete due to system failure or execution error, the dispatcher should store the contract state in persistant storage and resume the contract when conditions permit.

Table 1 summarizes the most interesting models and the control services required by each model. The only service necessary for all models is scheduling. System monitoring and dispatching are required by all models for reliable contracts. QoS monitoring services are necessary only for contracts requiring best effort or QoS guarantees. Brokers are required only by models requiring QoS guarantees. Arbitrators are needed only for multi external agent contracts.

Model	System Monitor	QoS Monitor	Dispatcher	Scheduler	Broker	Arbitrator
SUN				x		
SRN	x		x	x		
SRB	x	x	x	x		
SRQ	x	x	x	x	x	
MUN				x		x
MRN	x		x	x		x
MRB	x	x	x	x		x
MRQ	x	x	x	x	x	x

Table 1: Control agents necessary to support different contract models

We survey only models for contracts involving a single external agent and present a possible application for each model. The SUN model does not require service reliability or any QoS guarantees. We expect that few Intranets and contracts will only support this model. contracts to provide news services could use this model. An electronic commerce Intranet will support reliable and best effort, SRB, contracts. A metacomputing environment will most likely support SRB and possibly SRQ contracts.

Conclusions

In this paper we presented a taxonomy of the agents involved in a contract for network computing. We distinguish between external and internal network agents, execution and control agents, permanent and transient agents. Then we introduced models to capture the control, reliability and QoS requirements of a contract. Finally we discussed the architecture of an Intranet supporting several interesting models and presented the functions expected from the most important Intranet control agents.

Acknowledgments

This work was supported in part by the National Science Foundation through grants BIR-9301210 and MCR-9527131, by a grant from Intel Corporation and by the Scalable I/O Initiative.

Literature

[1] T. L. Casavant and J. G. Kuhl, “A Taxonomy of Scheduling General-Purpose Distributed Computing Systems”, *IEEE Trans. on Software Engineering,* 14(2), pp. 141-154, 1988.

[2] K. Mani Chandy, A. Rifkin, Paolo A.G. Sivilotti, J. Mandelson, M. Richardson, W. Tanaka and L. Weissman, “A World-Wide Distributed System Using Java and the Internet”, *IEEE International Symposion on High Performance Distributed Computing*, August 1996.

[3] K. Mani Chandy: “Caltech Infospheres Project Overview: Information Infrastructures for Task Forces”

[4] M.Lewis and A. Grimshaw, “The Core Legion Object Model”, *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, Los Alamitos, California, August 1996.

[5] M.G.Sirbu and D.C. Marinescu, “Bond - A Parallel Virtual Environment”, *Proceedings of HPCN Europe ‘96*, pp 722-728, Lecture Notes in Computer Science, Volume 1067, Springer Verlag, 1996

[6] M.G. Sirbu and D.C. Marinescu, “A Scheduling Expert Advisor for Heterogeneous Environments”, *Proceedings of the HCW’97 Heterogeneous Computing Workshop*, pp. 74-82, April 1997

[7] S.Franklin and A.Graesser, “Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents”, *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*, Springer-Verlag, 1996

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: “Hypertext Transfer Protocol -- HTTP/1.1” Internet Draft, HTTP Working Group, July 1997

[9] The Common Object Request Broker: Architecture and Specification, revision 2.1, August 1997, Object Management Group

[10] VisiBroker for Java - Reference Guide version 1.0, Visigenic Software Inc.

[11] Java IDL Guide - early access specification, Sun Microsystems 1997.