Design and Implementation of a Grid Architecture over an Agent-Based Framework

Christian Vecchiola, Alberto Grosso, Roberto Podestà, Antonio Boccalatte
DIST – University of Genoa
Via Opera Pia 13
16142, Genova, Italy
{christian, agrosso, ropode, nino}@dist.unige.it

ABSTRACT

Agent based programming presents several features appearing to be interesting for Grid and distributed computing needs. The typical environment required by Grid computing is complex, heterogeneous, and highly dynamic. The autonomous and flexible behavior provided by software agents meets various Grid requirements. In this paper we present the design and the implementation of a Grid architecture built over an agent based framework called AgentService. In this work we highlight the advantages in using the services of an agent oriented framework to develop a Grid application.

Keywords
Agent Mobility, Load Balancing Policy, Agent Framework, Grid computing

1. INTRODUCTION

Resource sharing through the Internet has become in the last years a paramount instrument for scientists, not only because it offers great advantages in distributed computing, but also because data sharing is becoming more and more useful in many scientific fields. Resources can be classified in three different groups: data, services, and computational power. By following this classification we can distinguish three types of grids [Fos01a]. Data Grids manage huge collections of geographically distributed data, which can be generated in many different ways, for example data streams are daily sent from satellites for weather forecast and climatic changes analysis; large collections of data generated from scientific experiments allow geographically distributed researchers to collaborate to the same research project. Service Grids provide services that could not be obtained from a single platform: for example streaming multimedia services or collaborative applications. Computational Grids provide the aggregate power of a collection of processors spread over the network as a unique, meta-computer.

In general, grid computing system are intended to replicate in the computing world the notion of a distribution grid fostered by utility networks such as the electrical power grid. In the vision of grid computing, computational power, memory, and disk space should be obtained "on demand" from a network of "suppliers", potentially belonging to the entire Internet.

In the last decade, distributed high performance computing has been built mainly on cluster computing systems where the communication among the different components of an application is performed using the message passing model implemented by systems such as MPI [MPI] and PVM [Gei94a]. Current trends in the grid community aim at providing frameworks not more strictly tied to the classical parallel computing programming model. However, is very hard to migrate this model to a dynamically changing environment such as the Internet, thus, in order to cope with the new challenges, a more structured, service and object oriented approach has to be adopted. The evolution toward a heterogeneous, dynamic, distributed over multiple domains environment has brought to the definition of the Open Grid Service Architecture [Fos02a] (OGSA), which proposes the convergence between grid computing and Web Services technologies in order to get over the classical parallel programming paradigm. The main, world-wide known Grid project, namely the Globus Toolkit [Fos05a], in its latest release implements the OGSA

.NET Technologies 2006
Copyright UNION Agency – Science Press,
Plzen, Czech Republic.

specification, and leverages on Web Services technologies and on the most widely known Internet standards. With this choice Globus is able to make available cluster-based high performance computing services to simple clients and end users without a specific parallel programming know-how too.

Moreover, starting from analogue considerations was conceived the Alchemi project [Luth05a]. Its authors aim at involve into the grid community the unused computational power provided by the almost ubiquitous Windows-based desktops. The Alchemi platform is a .Net-based framework providing the runtime machinery and the programming environment required to construct enterprise/desktop grids and to develop grid applications. Alchemi is able to interface with a Globus grid too, leveraging on a Web Services interface. Another effort trying to port the grid computing towards an object oriented programming model is the H20 [H20] project. This project provides a platform independent Java-based framework able to build meta-computing application leveraging on various remote method invocation protocol, such as SOAP, Java RMI, and TCP-based RPC [Kur03a].

It is our opinion that moving from the parallel computing programming model to a services and object oriented model, built on top of widely known technologies, can not be considered the last step of the Grid programming paradigm evolution. For example, Agent technology [Jen99a] and the agent programming model could be very useful to build virtual, highly dynamic, and distributed environment such as the context where typically operates a Grid framework. Agents are autonomous software entities with some level of intelligence; agents work better if they belong to a community such as a multi-agent system (MAS) [Wei99a]. Agents act in a distributed manner, cooperate, compete, and negotiate to solve a problem or to perform a task. These features make the agents an interesting technology to implement Grid infrastructures.

In this paper we present the design and the implementation of a Grid Computing architecture over an agent based framework. The Grid infrastructure has been built leveraging on the capabilities of the .Net based AgentService programming platform [Boc04a].

The paper is structured as follows: in section 2 we provide a brief overview on agent technology and multi-agent systems, and we describe the related synergy with grid computing; section 3 includes the description of AgentService programming platform; in section 4 we provide a detailed description of our agent-based Grid Computing architecture; section 5 shows a case study where our framework has been adopted; finally in section 6 we provide some concluding remarks, and we depict possible future works.

2. AGENT TECHNOLOGY AND GRID

A software agent is an autonomous software entity able to expose a flexible behavior. Flexibility is obtained by means of reactivity, pro-activity and social ability [Wei99a]. Reactivity is the ability to react to environmental changes in a timely fashion while pro-activity is the ability to show a goal directed behavior by taking the initiative. Social ability, that is the ability to interact with peers by means of cooperation, negotiation, and competition, is one of the most important features of agent oriented programming: agents do their best when they interoperate. Interaction is obtained by arranging agents in communities called multi-agent systems (MAS). MAS are generally decentralized open systems with distributed control and asynchronous computation: they provide a context for agents' activity with the definition of interaction and communication protocols. In addition they are scalable, fault-tolerant, reliable, and designed for reuse.

An abstract architecture specification of a generic multi-agent system has been proposed by the Foundation of Intelligent Physical Agents (FIPA), an international organization that promotes standards for agent technologies. The proposed architecture [FIP01a] is implemented by different multi-agent systems and has been taken as reference model in the comparison of different implementations of MAS.

Agents are reliable components to build flexible and fail safe systems, since autonomy and reactivity allow recovering from fault conditions. Agent and multi-agent technologies provide a promising approach to make Grid technologies smarter, more flexible, and adaptable. To support Grid computing, agents can offer different roles, be organized into dynamic groups, and be able to migrate between groups to support load balancing. Therefore, agents could play an important role in Grid computing, and Grid computing can offer useful test-beds for investigating Agent services. The social ability, the autonomous and flexible behavior could play an important role for the communication and the interaction with different nodes, for example, in exchanging information about the resources available on each node. The intrinsic nature of Agent technology, explicitly oriented to model high dynamic and complex systems [Woo99a], seems to be well suited to meet the Grid computing requirements. Moreover, the adoption agent technology could bring to Grid users and administrators more friendly and understandable interfaces to interact with the system. Some projects have already proved that the agent oriented approach could be adopted for Grid computing. The Agile Architecture and Autonomous Agent [Cao02a, Cao01a] (A4) is an agent based methodology for grid resource management. The computational power of the Grid is managed with a hierarchy of identical agents used to provide an abstraction of the system architecture. Each agent is able to cooperate with other agents to provide service advertisement and discovery to schedule applications that need to use grid resources. The Bond Agent System [BOND] is a FIPA project on top of which is possible to build agent based applications able to manage the state of the nodes and the coordination of a distributed system such a Grid [Kha03a].

3. THE AGENTSERVICE PROGRAMMING PLATFORM

AgentService [Boc04a] is a framework designed to develop multi-agent systems. It provides a class library to implement agents, an agent platform hosting multi-agent systems and a set of monitoring and design tools supporting either the development or the management of the MAS. The framework does not enforce particular agent architectures, but provides developers with a flexible agent model based on the concepts of knowledge and behavior. An agent is modeled, and implemented, as a software entity whose state is defined a set of knowledge objects, and whose activity is carried out by a set of concurrent tasks known as behavior objects. A knowledge object is a shared object containing related items which together define a unit of information. Knowledge objects can be shared among behaviors objects which model the different capabilities of an agent. AgentService comes with a set of extensions to the C# programming language that simplifies the development of agent applications. The AgentService object-oriented model is hidden by the APX [Vec03a], so that a clear agent-oriented interface is offered to the developers with slight changes to the C# syntax.

The platform provides a complete environment to execute agent instances which rely on the advanced services of the platform: repository, communication, and directory services. Some of these services become strategic when platform instances constitute the nodes of a computing grid. In particular directory and communication services are discussed in detail.

Directory Service

AgentService has been designed following the architectural specifications provided by FIPA which states that a set of basic services are required on each agent platform. These are implemented as agents and are:

- Agent Management Service (AMS) - supervisor and controller of the platform services;
- Directory Facilitator (DF) - providing yellow pages service;
- Message Transport Service (MTS) - managing communication service.

Directory services are fundamental in dynamic and distributed environments due to the fact that a single entity needs to know if, when, and where a specific service is available. For these reasons DF is a compulsory component for an agent platform. In AgentService each platform provides a directory service to agents. By registering to the DF agents can specify the services they offer and their communication profile. Directory Facilitator agents scattered on AgentService platforms can join together to form a federation, hence if an agent registers to the local DF, it becomes visible, and advertises its services, to all the platforms of the federation. When deregistration occurs the information is spread on all the nodes of the federation. DF agents maintain a distributed database of all the services available on the federation: each agent by interacting with the local DF gets access to an entire net of services. DF agents according to the service profile advertise it on all the nodes of the federation or just to a subset of it. The ability of controlling the advertising policy allows a better use of the network resource.

Communication Infrastructure

A dedicated agent, the MTS (Message Transport Service) is responsible of managing the platform messaging subsystem. The messaging subsystem is implemented within a module and by default AgentService provides a communication service based on message exchange and conversations (connected communication between two agents). The ability of changing the implementation and the communication channel among platform nodes in a transparent manner for agents is remarkable advantage of this architecture: the implementation of the module is hidden to the MTS, and then to the agents, which interacts with the module through the IMessagingModule interface. The messaging module creates and maintains a specific message queue for each agent hosted in the platform and can choose the best technology solution to store this information (a database, a file system, or a message queuing service). Messages exchanged among agents are compliant to the FIPA specifications and need to contain only serializable items, since messages may trespass the boundary of the single machine. The default messaging module provided with the AgentService installation comes with two fundamental services: conversations and inter-platform message dispatching. Conversations are connected message exchange services and provide a useful abstraction to model interaction protocols. Inter-platform message dispatching allows the community of agents to extend beyond the single platform instance boundaries. The communication among different AgentService installations is based on the Web Service infrastructure provided by .NET framework. Hence soap messages are exchanged among platforms and a specific format of the xml content is defined by AgentService to ensure the secure and correct delivery of the messages. The .NET automatic serialization process for the soap messages has been customized to allow the serialization of agent messages and to decrease the amount of the transferred data without loss of information.

Additional Services

The platform has been designed to be an extensible software environment: the community of agents hosted in the federation of platforms evolves and additional features may be required when the system is installed. Hence the ability to extend the proper capabilities becomes a requirement. The architecture of the platform allows third party modules to be integrated into the platform core and to offer services to either the other modules or the agents. By using this technique the platform has been extended by an FTP service available to all the other platform components. Agents and other modules can require a folder space or just send files by using the service as a simple FTP client.

The directory service, the communication infrastructure and additional services, together with a set of dedicated agents constitute the core of the grid infrastructure provided with AgentService.

4. DESIGN OF A GRID INFRASTRUCTURE OVER THE AGENTSERVICE PLATFORM

The elements defining the grid infrastructure are agents, platform components, and additional services. Agents encapsulate the logic of the system while platform components and additional services maintain its structure. This organization is replicated on each installation of the platforms participating in the grid.

Figure 1 gives an overview of the entire system. The federation of the platforms defines the boundary of the grid. The structure of the systems is dynamic since AgentService instances can dynamically join the federation by sending a message to the agent managing one of the nodes. In the same way nodes can detach from the system. This is a fundamental feature for grid systems that are dynamic by nature. According to the configuration of the node each platform can act as a computational node, provide access to the system, or perform both the two roles.

The System's Logic: the Agents

The logic of the system is composed by a community of specific agents deployed on each installation of the AgentService platform. In this section we will describe the tasks delivered to each agent and how they take advantage of the services offered by the platform to deploy and to manage the infrastructure of the computational Grid.

Each node which is part of the grid infrastructure is equipped with an installation of the AgentService platform. On each node the platform hosts the following agents:

- NodeManager: the NodeManager is the maintainer of the node, it coordinates all the activities required to implement the grid service. The NodeManager maintains a registry of the platforms which constitute the computational grid and manages the dynamic registration of platform instances. The NodeManager is responsible of assigning a task to a specific node by looking at the topology of the grid, at the computational load of each node, and at the services offered by that node.

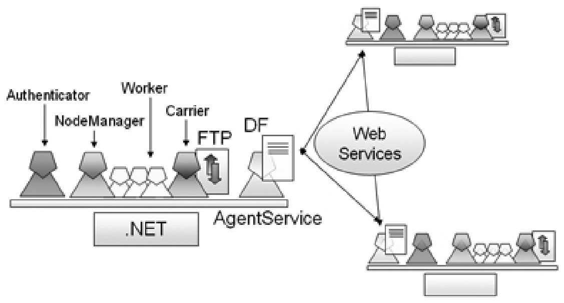

Figure 1. A graphical overview of the Grid Architecture based over the AgentService Framework

- Carrier: the Carrier agent is responsible of transferring on the selected node of the grid all the resources required to perform the task. The Carrier relies on the file transfer service offered by the platform, by which it delivers to the selected node the object code containing the task and all the related input or data files.

- Authenticator: an instance of the Authenticator agent manages the security of the node; it maintains a registry of user profiles, checks the user credentials when a task is submitted to the grid, and applies the security policies defined in the user management module.
- Worker: multiple instances of the Worker agent are hosted on each node and take care of tasks execution. On the selected node, the NodeManager contacts the worker agent every time a new task needs to be executed; the worker agent sets up the computing environment required for the task, executes the task, and eventually communicated the results. The NodeManager agent can limit the maximum number of concurrent Worker agent instances in order to control the computational load of the node. Worker agents can perform many tasks concurrently thanks to agent model adopted by AgentService. The tasks partition criteria among worker agents can be defined as configuration parameters of the node or dynamically decided by the NodeManager; a simple selection criterion could be dividing the tasks according to the permission of the users they belong to.

Tasks are submitted to the grid and NodeManager agents cooperate to identify the candidate node on which the task will be executed. Since cooperation, negotiation, and competition are natural activities in multi-agent systems this functionality is naturally obtained by using the agent oriented approach. In the same way localization of services and coordination within a single node are obtained with less effort.

The Grid Structure

The community of agents that is distributed on the nodes constituting the grid gives a high level view of the entire grid. The implementation of the infrastructure strongly relies on the core services of the platform. In particular, communication services, file transfer, and localization. These features are respectively implemented by using the messaging subsystem, the FTP service, and the DF agents spread on each node.

The messaging subsystem is one of the core elements of both the multi-agent system and the grid infrastructure implemented on it. Software agents interact with peers by exchanging messages; hence the coordination of the elements defined in the logical layer is based on the messaging subsystem. The ability to communicate with peers hosted on other nodes is a requirement to distribute computation; hence, the installation of AgentService has been customized with a messaging module that uses the web services technology to deliver messages on other platforms. The use of web services provides a solid, well known standard allowing interoperability and integration with other applications. Agent messages are required to be serializable but not to be represented by using a SOAP message. The platform replaces the default XML serialization provided by the .NET framework with a custom technique that reduces the body of the SOAP message and allows the transport of any serializable managed type. The messaging module attaches the description of the type to the binary serialization of each item in the agent message; the binary instance is encoded into a base64 string and transmitted as an attribute of the XML element representing the item. On the target node each item is reconstructed according to the type information attached to the item: the full name of the type, its assembly name, and the public key token of the assembly are used to de-serialize the instance into the original object. This solution speeds up the transmission of any complex object via web services, avoids type mismatch, and is completely transparent to developers which are not required to provide an XML serializer for every type they define.

The ability to transfer objects among platform nodes is a requirement for distributing the computation. The messaging subsystem provides a simple way to transport messages but it cannot handle efficiently the transfer of large amount of data. Moreover, the communication infrastructure has been designed to send .NET instances and not for large files. For this reason, the installation of AgentService has been enriched with an additional module that handles the FTP protocol. The module integrates into the platform and provides this feature as service. The FTP service can be exploited either by software agents or platform modules and it is mainly used to move on the target node all the assemblies containing the code executing the task and the required data files. Modules and software agents can dynamically check the availability of the service and eventually require a personal folder or just submit a file to transfer. When files are uploaded to the server the owner of the folder is notified about the transfer. In this case the FTP service is mainly used by the Carrier agent who is responsible of transferring the assemblies containing the task to be executed on the target node. Carrier agents ask for a personal folder to the FTP service and the FTP service creates the corresponding directory in the root folder of the FTP server. When a task is moved to a node of the grid the Carrier agent on the source platform instruct the FTP service to upload the file on the target platform. When the upload is finished the FTP service of the target platform notifies the Carrier agent about the transferred files. The same interaction pattern is used by modules if they need to send or receive files.

Localization and discovery of services play an important role in distributed systems. The ability to discover agents and the services they offer is a requirement for agent communities which are dynamic by definition. These are requirements for Grid systems too: nodes should be able to obtain information about other nodes in order to distribute the load. Within AgentService a distributed directory service is responsible of advertising and retrieving services available in the multi-agent system. Directory Facilitator agents constitute a federation sharing all the information about published services. DF agents provide information to NodeManager and Carrier agents: the first ones query the local DF in order to know all the other NodeManager agents and set up the topology of the grid; the second ones look for Carrier agents when they need to transfer files on a selected node. DF agents are also useful for connecting agents within a single node: each of the previously defined agents register its service profile to the local DF. Directory Facilitator agents can be instructed for a local search: in this way the agents defining the logical layer of the grid connect each other.

Many of the elements constituting the infrastructure of the grid are provided by the environment hosting the agent. These elements are commonly required by the agents to perform their activities; hence the use of a multi-agent system for grid computing can strongly simplify the development of grid system. In addition, the modular architecture of the AgentService platform and its natural extensibility allows the simple implementation of the missing features as in the case of the FTP service.

5. CASE STUDY

A common computing task submitted to the grid can be taken as a case study since it is useful to describe the interaction among the agents modeling the logical layer of the grid and their connection with system components.

Users that want to submit a job to the grid have to contact those nodes which are configured as access points to the grid. These nodes are the starting point of the entire process. The user authenticates by sending a message containing tis credentials to the Authenticator agent of the access point. Installations of the AgentService platforma provide a communication channel that can be used by GUIs or web applications for remote management and access: the common scenario involves a web application connecting to the access point through a web browser submitting a task by uploading all the required files. The web application connects to the platform with the credentials provided by the user and queries the DF for the Authenticator agent which checks the user permissions and validates the request of the user. The Authenticator agent sends a message to the NodeManager agent of the same platform which selects the best node of the grid according to:

- the user profile;
- the type of task to execute;
- the availability of processor cycles on each node.

In order to select the best node NodeManager interacts with the other NodeManager agents hosted on the other platforms. The NodeManager agents maintain updated the state of the entire grid by exchanging messages when interesting events occur (a task is finished, a task is started, a task has been aborted); hence, each NodeManager agent is always aware of the status of the grid.

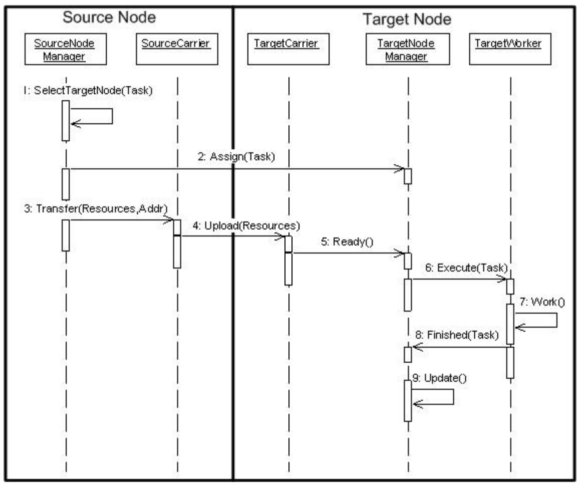

Figure 2. Sequence diagram describing the protocol for task execution

Once the node has been selected the local NodeManager is contacted to start the task. The target node could require additional resources to perform the task and in that case the NodeManager agent instructs the local Carrier agent to accomplish the transfer on the target site. The local Carrier agent by querying the DF looks for the remote Carrier agent and then sets up the transfer by using the local FTP service. On transfer completion the Carrier agent on the selected node notifies the local NodeManager that all the resources required to perform the task are available. This is the final step of the activation process: the NodeManager agent according to the computational load of the node requires a new Worker agent or submits the work request to an active Worker agent. The number of active Worker agent can change on each node and the NodeManager itself can dynamically decide the best policy to apply. Figure 2 depicts the sequence diagram describing task execution after the credential of the user have validated.

The Worker agents picks up a new work request inspects the information describing the task to execute and by means of reflection creates a new instance of the type defining the tasks, starts its activity by using a configuration files transmitted along with the resources. Assemblies containing the tasks can be cached on the nodes in the platform storage and useless transfers can be avoided. The types must implement the following interface:

```
interface ITask
{
  bool IsReusable { get; }
  Exception Error { get; }
  bool Prepare(string configFile);
  void Execute();
  bool Abort();
  bool Dispose();
}
```

In order to execute a task the Worker agent creates an instance of the required type and invokes the Prepare method that configures the task to execute. If the method returns true the task will be executed by invoking the Execute method and upon completion a call to Dispose finalizes the execution and eventually communicates the results. Exceptions occurred during execution are obtained by looking at the Error property while, while IsReusable is true if the same instance can be used to perform many tasks of the same type in sequence. Two additional interfaces are provided to make tasks execution more flexible: IControllableTask and IIterativeTask. The first one adds facilities to control task execution with a pause-resume pattern while the second one allows the execution of tasks one step at time.

When the task is finished, the Worker agent notifies the NodeManager about completion which update the status of the grid.

6. CONCLUDING REMARKS AND FUTURE WORKS

Agent technology seems an interesting solution to implement distributed and dynamic computational environments: agents confer a certain degree of autonomy to the system components and simplify the creation of dynamic relations among them. Hence, the use of such technology in the field of grid computing is a reasonable and interesting approach. This paper has presented the design and the implementation of an infrastructure for grid computing which relies on agent technology and takes advantage of the AgentService framework. The community of agents defines the logic of the system while the extensible core of the agent platform implements the low level services required by a Grid architecture. This approach has two main advantages:

- the coordination and task distribution policies can rely on the interaction capabilities of agents: they are high level system components which naturally embed negotiation, competition and cooperation capabilities;
- the default services provided by multi-agent system meet typical grid computing requirements; hence the use of a modular and extensible multi-agent system, like AgentService, as a backbone simplifies and improves the efficiency in the Grid architecture development.

The structure of the system is based on a net of platform instances connected together by using the web services technology. Web services are used only for communication and AgentService implements custom technique which allows the transfer of any .NET serializable and complex object, keeps the SOAP packet small, and speeds up the transfer. The use of web services could lead to possible performance bottlenecks but message exchange among agents should have a small cost if compared to the time required to perform tasks submitted to the grid. In addition, AgentService uses web services only for communication and has been enriched with an FTP service that is used to move object code and data files among node.

The architecture described in this paper is specifically designed for computational Grids, but the underlying model can be applied also to other types of grids. A possible extension of the presented architecture could be the ability to move agents which are performing a task in order to apply load balancing policies. This service could be provided by adding a mobility module in order to provide a task migration service. This module allows agent instances to cross the platform boundaries and move among AgentService platform instances.

7. REFERENCES

[Fos01a] Foster, I., Kesselman, C., and Tuecke, S. The Anatomy of the Grid. Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 2001

[MPI] Message Passing Interface Forum. Message Passing Interface, documentation available on line at www.mpi-forum.org

[Gei94a] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Mancheck, B., and Sunderam, V. PVM: Parallel Virtual Machine a User’s Guide and

Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, 1994

[Fos02a] Foster, I., Kesselman, C., Nick J., Tuecke S., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Global Grid Forum, June 22, 2002

[Fos05a] Foster, I., Globus Toolkit Version 4: Software for Service-Oriented Systems, IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005

[Lut05a] Luther, A., Buyya, R., Ranjan, and R.,Venugopal, S., Alchemi: A .NET-Based Enterprise Grid Computing System, Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), June 27-30, 2005, Las Vegas, USA.

[H20] H2O Project, http://www.mathcs.emory.edu/dcl/h2o/

[Kur03a] Kurzyniec, D., Wrzosek, T., Sunderam, V., and Slominski, A.. RMIX: A Multiprotocol RMI Framework for Java. In Proc. of the International Parallel and Distributed Processing Symposium (IPDPS'03), pages 140-146, Nice, France, 2003

[Jen99a] Jennings, N.R., and Wooldridge, M., Agent-Oriented Software Engineering, Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99), 1999

[Wei99a] Weiss, G., Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, 1999

[Boc04a] Boccalatte, A., Gozzi, A., and Grosso, A., Una Piattaforma per lo Sviluppo di Applicazioni Multi-Agente, WOA 2003: dagli oggetti agli agenti – sistemi intelligenti e computazione pervasiva, Villa Simius, Italy, September 2003

[FIP01a] FIPA Abstract Architecture Specification, http://www.fipa.org/specs/fipa00001/

[Woo99a] Wooldridge, M., Intelligent Agents, in Multi-agent Systems – A Modern Approach to Distributed Artificial Intelligence, G. Weiss Ed., Cambridge, MA, 1999, pp. 27-78

[Cao02a] Cao, J., Spooner, D. P., Turner, J. D., Jarvis, S. A., Kerbyson, D. J., Saini, S., and Nudd, G. R., Agent-Based Resource Management for Grid Computing, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)

[Cao01a] Cao, J., Kerbyson, D. J., and Nudd, G. R., Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing, Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01), Brisbane, Australia, May 2001

[BON] BOND Project, http://bond.cs.ucf.edu/

[Kha03a] Khan, M.A., Vaithianathan, S.K., Sivoncic, K., and Boloni, L. Towards an Agent Framework For Grid Computing, CIPC-03 Second International Advanced Research Workshop on Concurrent Information Processing and Computing, Sinaia, Romania, 2003

[Boc04a] Boccalatte, A., Gozzi, A., Grosso, A., and Vecchiola, C. AgentService. The Sixteenth International Conference on Software Engineering and Knowledge Engeneering (SEKE'04), Banff Centre, Banff, Alberta, Canada 20-24 June 2004

[Vec03a] Vecchiola, C., Coccoli, M., and Boccalatte, A. Agent Programming Extensions relying on a component oriented infrastructure, Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration (IRI - 2003), Oct. 26-29, Las Vegas, NV, 2003.