An Object-Oriented Framework for Building Software Agents

Ruy L. Milidiú
milidiu@inf.puc-rio.br

Carlos José Pereira de Lucena
lucena@inf.puc-rio.br

José Alberto Rodrigues Pereira Sardinha
sardinha@inf.puc-rio.br

Paula C. Ribeiro
pclark@inf.puc-rio.br

PUC-RioInf.MCC06/02 April, 2002

Abstract: Agent technology is a new approach of Distributed Artificial Intelligence to implement autonomous entities driven by beliefs, goals, capabilities and plans, and other agency properties such as adaptation, interaction, and mobility. Software agents are the focus of considerable research in the artificial intelligence community, but not much has been done in the field of software engineering. In this paper, we present an object-oriented framework for building software agents in a distributed environment. The design of the framework also allows an easy mapping of the models developed in the analysis and design phase of the Gaia Methodology to object-oriented code. We believe that object-oriented framework technology can reduce not only the development time but also the complexity of implementing multi-agent systems. We present an instantiated application that uses this framework to illustrate an implementation.

Keywords: OO Frameworks, Software Agents, Gaia Methodology.

Resumo: A tecnologia de Agentes de Software é uma nova abordagem da Inteligência Artificial Distribuída para implementar entidades autônomas dirigidas por crenças, metas, habilidades e planos, e outras propriedades de agência tais como interação, adaptação e mobilidade. Agentes de Software é o foco de vários trabalhos na comunidade de inteligência artificial, mas muito pouco vem sendo feito na área de engenharia de software. Neste artigo, apresentamos um *framework* orientado a objetos para desenvolver agentes de software em um ambiente distribuído. O *design* do *framework* permite um mapeamento direto dos modelos desenvolvidos pela Metodologia Gaia para código orientado a objetos. Acreditamos que a tecnologia de *frameworks* é capaz de reduzir o tempo de desenvolvimento e a complexidade de implementação de um sistema multi-agentes. Apresentamos uma aplicação que utiliza o *framework* para ilustrar uma implementação.

Palavras-Chave: *Frameworks* OO, Agentes de Software, Metodologia Gaia.

1. Introduction

Multi-Agent Systems [1] [2] is a new technology that has been recently used in many simulators and intelligent systems to help humans perform several tasks. To achieve the system's goal, agents have to react to events, define strategies, interact, and participate in organizations. However, Software Agents have gained great importance for both academic and commercial applications with the advent of the Internet. Applications for the Internet are easily modeled with agents, mainly because of the distributed approach. We believe that many other applications based on this technology are still to be built to help leverage the use of the Internet.

We define Software Agents [3] as an autonomous entity driven by beliefs, goals, capabilities, plans and a number of agency properties, such as autonomy, adaptation, interaction, learning and mobility. Although we recognize that the object-oriented paradigm has some flaws [3][4][5] related to design and implementation of multi-agent systems, we also believe that it is still the most practical programming language to implement the agent technology.

Our object-oriented framework [6] implements a communication infrastructure for agents over a network, and uses some hot spots [6] in order to implement the agent's beliefs, goals, capabilities, plans and some agency properties, such as autonomy, adaptation, interaction, and learning. This framework also permits an easy mapping of the models developed by the analysis and design phase of the Gaia Methodology [7] to object-oriented code.

Kendall et all [8] also propose an agent framework that is an architectural pattern in layers. In this architecture, an agent is composed of seven layers, such as the layer of sensors that are responsible for detecting changes in the environment. This pattern permits the modeling of both simple and complex agents. In [9], this agent design pattern is criticized for being too general. Consequently, some difficulties arise in the maintenance process and evolution of the system. The process of modeling in layers does not permit an easy removal of a layer when changes are needed. Adjacent layers to the removed layer normally have to suffer changes in order to adapt the system.

We used IBM's TSpace [10] software to implement the communication infrastructure. IBM TSpace is a reflective tuple space architecture [11] that provides support to all basic associative blackboard [2] operations (read, write, and take). TSpace can also be programmed to react to specific stimuli, and we used this feature to enable the exchange of messages between software agents. In fact, our communication infrastructure is a layer over TSpace that provides blackboard and message passing communication for agents.

The agent framework is called MAS Framework, and it is used in three projects of the TecComm Group - Software Engineering Laboratory at PUC-Rio. The first development [12][13] uses agents to build a tool for creating offerings automatically in a retail market. The second tool [14] uses software agents to encourage people to participate in Consuming Groups, and the third project [15] uses agents for negotiation purposes in a virtual marketplace. All projects were able to re-use the same code and consequently reduce the development time and effort.

In Section 2, the MAS Framework is presented in detail and in Section 3 we describe how to instantiate an application that uses this framework. In Section 4, we present a case study of a project that uses the framework.

2. The MAS Framework

The main goal of the MAS Framework is to diminish the development time and reduce the complexity of implementing Software Agents. The design of this framework is inspired on the Gaia Methodology, and it can introduce an easy implementation of all the models developed by the analysis and design phase.

Gaia is a methodology for building models in the analysis and design phase. It is applicable to a wide range of multi-agent systems, and also deals with both the macro-level (societal) and the micro-level (agent) aspects of the system. It considers the system as a computational organization consisting of various interacting roles.

Gaia allows an analyst to go systematically from a statement of requirements to a design of models that can be coded, for example, with the MAS Framework. The first two models that are built in Gaia are the Roles model and the Interaction Model. The Roles model identifies the key roles in the system or an entity's expected function. However, there are many dependencies and relationships between the various roles in an agent organization. These dependencies and relationships originate the interaction model, which is responsible for describing the various interaction situations between the agent's roles.

Both the Roles model and the Interaction model derive three other models: Agent model, Services model, and Acquaintance model. The Agent model describes an agent type with set of roles (as identified in the Roles model). The Services model is to identify the services associated with each agent role, or we can understand as the agent's functions. The Acquaintance model defines the communication links that exist between agent types.

The MAS Framework is composed of one abstract class – *Agent*, two final classes – *ProcessMessageThread* and *AgentCommunicationLayer* and four interfaces – *AgentMessage, AgentBlackBoardInfo, InteractionProtocol and AgentInterface*. All of these classes have been developed in Java, and in the following paragraphs we will describe in detail each class and interface.

The Agent model in Gaia has a direct mapping with the MAS Framework. Consequently, every software agent modeled in the Agent model has to instantiate an Agent from the MAS Framework. This instantiation includes the specialization of the abstract class *Agent*, the implementation of the interfaces (*AgentMessage, AgentBlackBoardInfo, InteractionProtocol and AgentInterface*), and the use of the objects *ProcessMessageThread* and *AgentCommunicationLayer*.

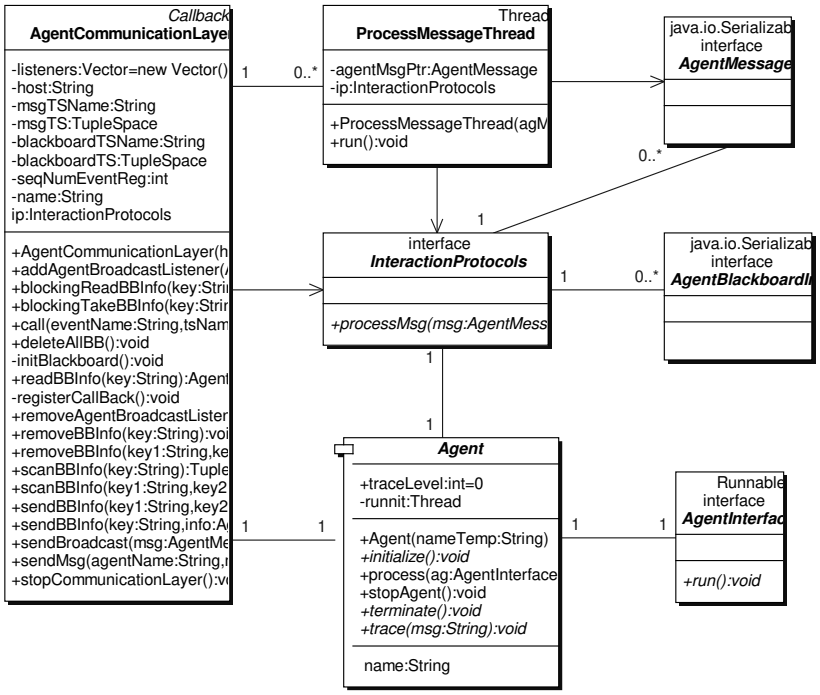

Figure 1 – MAS Framework

Every service in the Services model will be implemented as a method. The services that are derived from activities in the Roles model will be coded as methods in the specialized class of *Agent*. Analogously, the services that are derived from protocols are coded in the class that implements the interface *InteractionProtocol*. Furthermore, the inputs and outputs of the Services model and the permissions of the Roles model are coded as attributes of the specialized class of *Agent*. The pre-conditions and post-conditions of each service are also coded in the specialized class of *Agent*, and it is possible that some of these pre-conditions and post-conditions have to be implemented using monitors.

The Interaction model in Gaia is very useful because it represents the interactions between the roles in the Roles model. As the roles are mapped to agents in the Agents model, the interactions also represent interactions between agents. Consequently, there is a direct mapping between the interaction models and the sequence diagrams in UML. The sequence diagrams will use the methods coded in the specialized class of *Agent* and the class that implements the interface *InteractionProtocol*.

Agent is an abstract class, and the subclass that inherits it implements the "private" actions or activities [7] of the Software Agent's roles. These private actions do not depend on any interaction with other agents in order to accomplish specific tasks. The developer is obliged to implement methods related to the startup code (method *initialize*), ending code (method *terminate*), and display of messages (method *trace*). The methods *process* and *stopAgent* are to start and stop the agent. The attribute *name* specifies the agent's name in the system, and due to implementation details it has to be unique.

The *AgentInterface* is responsible for making the subclass inherited by *Agent* into a thread. This subclass implements a method called *run*, and this method is responsible for starting the agent's private activities.

The *InteraactionProtocols* is an interface of a class that will define the way a software agent can interact with other agents in the society. All the code related with interaction is placed in this class. The implemented class also requires the implementation of a method called *processMsg*. This method is called every time a new message is received from another agent.

The *ProcessMessageThread* is in charge of processing messages received by agents. In fact, it creates a new thread for every incoming message, which will automatically call the abstract method *processMsg*. The *AgentMessage* is an interface used by the class that specifies the message format, and *AgentBlackboardInfo* is an interface used by the class that specifies the blackboard message format. Thus, all blackboard information and messages in the system must implement these interfaces.

The *AgentCommunicationLayer* is a class that implements the entire communication infrastructure needed for agents to interact in a distributed system over a network. This infrastructure is a layer over IBM Tspaces. A summarized description of every method is displayed below:

addAgentBroadcastListener - Method that adds an agent to the broadcast listener group;

blockingReadBBInfo - This method is used for reading information in the MAS blackboard. When the agent issues a *blockingReadBBInfo* call, and the data is not yet there on the MAS blackboard, the application blocks the call until an answer is returned. When the information arrives on the MAS blackboard that matches the *blockingReadBBInfo* query, it is sent to the Agent and it resumes;

blockingTakeBBInfo - This method is used for removing information in the MAS blackboard. When the agent issues a *blockingTakeBBInfo* call, and the data is not yet there on the MAS blackboard, the application blocks the call until an answer is returned. When the information arrives on the MAS blackboard that matches the *blockingTakeBBInfo* query, it is sent to the Agent and it resumes;

deleteAllBB - Method used for deleting all information in the MAS blackboard;

readBBInfo - This method is used for reading information in the MAS blackboard;

removeAgentBroadcastListener - Method that removes an agent from the broadcast listener group;

removeBBInfo - This method is used for removing information in the MAS blackboard. No blocking service is offered in this call;

scanBBInfo - This method is used for querying information in the MAS blackboard;

sendBBInfo - This method is used for posting information in the MAS blackboard;

sendBroadcast - This method is used for sending a broadcast message for the registered agents;

sendMsg - This method is used for sending a message to an agent;

stopCommunicationLayer - This method is used for ending the communication layer.

Object oriented framework design can be divided into two parts [16]: the kernel subsystem and the hot spot subsystem. The kernel subsystem design is common to all instantiated applications, and in the MAS Framework the classes *AgentCommunicationLayer* and *ProcessMessageThread* represent it. Hot spot design describes the different characteristic of each instantiated application. In our framework the hot spots are the classes *Agent, InteractionProtocol, AgentMessage, AgentBlackboardInfo* and *AgentInterface*.

3. How to Instantiate the MAS Framework

In this section we will describe the instantiation process of the framework, and show the interdependence of the classes. It is important to have an IBM TSpace server running in your network or local machine. The configuration of the IBM TSpace server can be found in [10].

The first class to be extended is *Agent*, and the subclass that inherits it will implement the "private" actions or activities of the software agent's roles. This subclass shall also implement the interface *AgentInterface*, and write code for the methods *initialize*, *run*, *terminate*, and *trace*. When an activity ends up in an interaction protocol, it is necessary to have a reference to the class that implements the *InteractionProtocols*.

A class has to implement the interface *InteractionProtocols*, and all the code related with interaction is placed here. The method *processMsg* needs to have code that interprets an incoming message, and a reference to *AgentCommunicationLayer* is required in order to implement interaction through the communication infrastructure. In Figure 2, we present a simple instantiation of the MAS Framework.

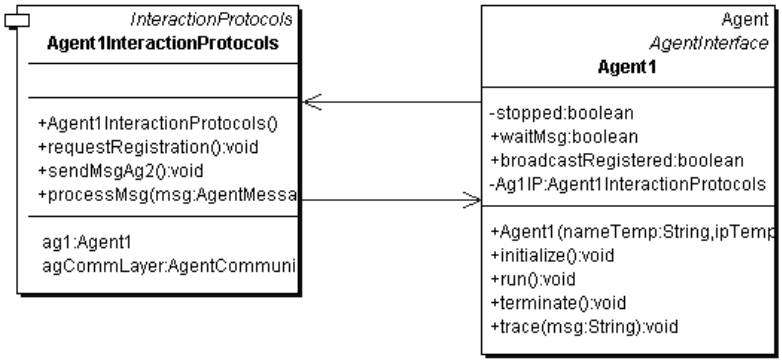

Figure 2 – An instantiated application

If the implemented software agent uses message passing in order to communicate, a class that specifies the message format shall implement the interface *AgentMessage*. If the agent also uses a blackboard for communication, another class that specifies the message format shall implement the interface *AgentBlackboardInfo*.

4. The Babilonia application – A case study

The Babilonia application is a virtual marketplace for buying and selling goods. It also implements an automatic negotiation of the buyer and the seller with software agents. The system uses certification of buyers, sellers and goods to ensure the security of the transactions.

Our multi-agent system is modeled using an organizational design with the following roles: buyers, sellers and certification agent. The buyer is responsible for registering a profile that shall be used by the system to find goods. This profile shall include some characteristics of the item such as brand and model. Every item that matches the registered profile is sent to the buyer, which shall decide whether or not to start the negotiation process. If the negotiation is authorized, a software agent conducts such work. However, the agent requires some parameters: initial offering price, maximum price offered, and an incremental value used in the counter-proposal.

The seller also is responsible for registering a profile that will be used by the system to find goods. An agent is created for each profile, and to conduct the automatic negotiation some parameters are required: initial offering price, minimum price offered, and a decrement value used in the counter-proposal. The seller also shall inform the system if a certification of the buyer is required.

The certification agent is responsible for certifying the seller, the buyer and the goods involved in a transaction. This agent has to look up information in public databases outside of the system to be able to certify the whole transaction. The agent may eventually require help from a user.

The system waits for the registering of profiles, and the first step involves the matching of a buyer profile and a seller profile. If a match is found and both parties agree to start to negotiate, software agents start the automatic negotiation. If both agents agree to a price for the item, everyone is notified that the transaction has ended. The system is not responsible for delivering the item, but sends a notification to both parties with each other's personal information. In Figure 3, we present an illustration of an instantiated buyer agent.

When we use Gaia, the first step we must take is to transform a problem into the Roles model and the Interaction model. In this paper we will only describe the modeling process that created the buyer agent. However, a complete modeling and mapping to the MAS Framework can be found in [15]. It is important to emphasize that the Roles model and the Interaction model should not be constructed separately. The models must be developed together in order to maintain the consistency of the process.

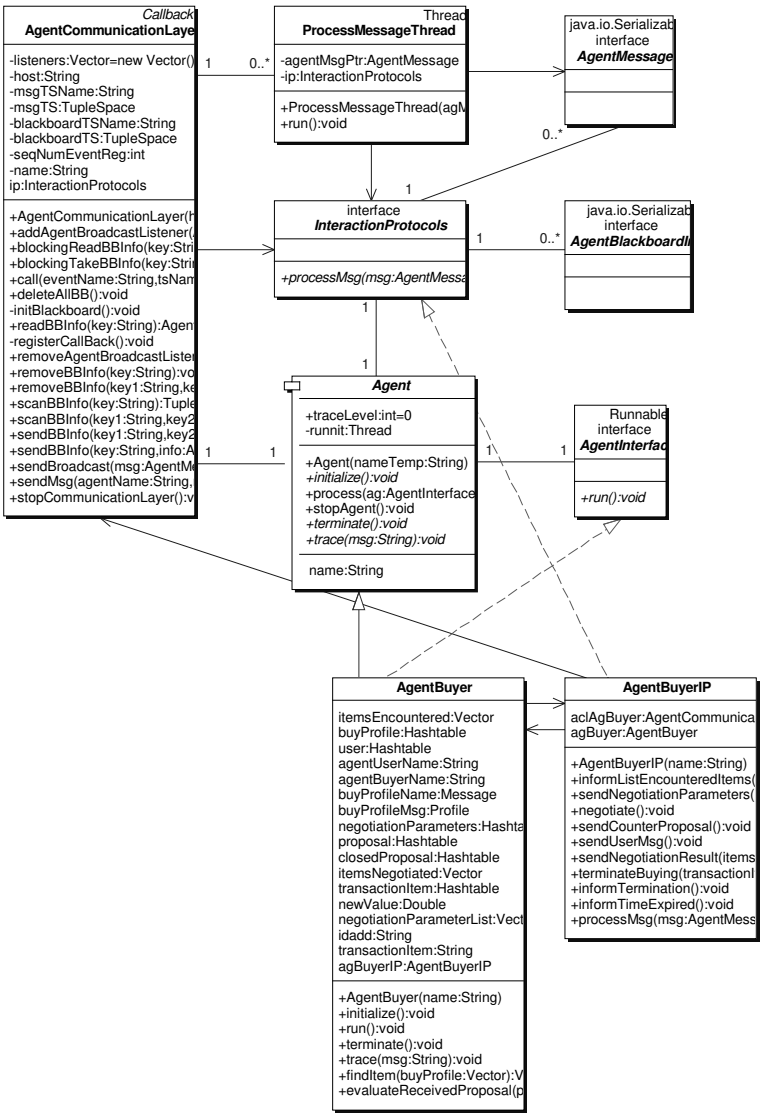

Figure 3 – An instantiated buyer agent

One of the roles of the Babilonia application is called Buyer, and it is responsible for finding items in the environment and negotiating them for the best possible price. A role indicates a presence of an entity that may turn into a software agent, but this is not necessarily true. In this phase, it is important to notice that we do not have to worry if a role will become an agent or not. Another important step in the Roles model is to identify a role's functions or responsibilities. Every responsibility is composed of protocols and activities. The protocols represent functions that need interaction with other roles, and the activities are internal functions of a role. The Role model of the buyer is found in Figure 4.

Role Schema: Buyer

Description: Responsible for finding an item (of a specific profile), and mediate the buying process.

Protocols and Activities:
FindItem, InformItemListEncountered, SengNegotiationMessage, EvaluateReceivedProposal, SendNegotiationResult, TerminateBuying, InformTermination, InformTimeExpired, Terminate

Permissions:

Reads	**informs** *userAssistant*
	buyProfile
	selectedItems
	negotiationParameters
	transacationItem
	proposal
Generates	*itemsEncountered*
	itemsNegotiated
	buyer
	lifeTime
changes	*proposal*

Responsabilities:
Liveness:

BUYER = START . (FIND . [NEGOTIATE . BUY] | FIND)$^{\omega}$. STOP
START = FindItem
FIND = InformItemListEncountered
NEGOTIATE = (SengNegotiationMessage. EvaluateReceivedProposal) $^{\omega}$
BUY = SendNegotiationResult . TerminateBuying
STOP = (InformTermination | InformTimeExpired) . Terminate

Safety:

lifetime < lifeLimit

Figure 4 – The Buyer's Role Model

Every protocol in the Roles model will participate in an interaction that is explicitly modeled in the Interaction model. The main purpose of this model is to indicate how every protocol will execute, and consequently point out the roles that participate in this execution. More details about the Interaction model of the Babilonia application can be found in [15].

The Agent model is simple and formalizes something that usually is already known in the Roles model – which roles are associated with a software agent. In our system, the buyer agent has the attribution of only one role – Buyer. Another model that derives

from the Roles model and Interaction model is the Services model, and it is responsible for identifying the services offered from every agent. The Service model of the buyer agent can be found in figure 5.

Service	**Inputs**	**Outputs**	**Pre-condition**	**Post-condition**
find item	*buyProfile*	*ItemsEncountered*	true	True
inform list encountered items	*itemsEncountered*	*ItemsEncountered*	true	True
send negotiation message	*proposal*	*proposal*	Negotiating = false	Negotiating = true
evaluate received proposal	*proposal*	*proposal*	true	True
send negotiation result	*itemsNegotiated*	*ItemsNegotiated*	true	Negotiating = false
terminate buying	*transacationItem*	*buyer*	true	True
inform termination				
inform time expired				
terminate				

Figure 5 – The Buyer's Service Model

Every service in the Buyer's Services model is implemented as a method. The services that are derived from activities in the Roles model will be coded as methods in the *BuyerAgent* class. Similarly, the services that are derived from protocols are coded in the *BuyerAgentIP* class. In addition, the inputs and outputs of the Services model and the permissions of the Roles model are coded as attributes of the specialized class of *Agent*. The pre-conditions and post-conditions of each service are also coded in the *BuyerAgent*.

5. Final Comments

Although the Gaia methodology is a very useful tool for the analysis and design phase, the MAS Framework can also be used with other methodologies. However, we encourage a developer to think of building a multi-agent system as part of the process of an organizational design. Consequently, our agent-based system can be seen as an artificial "society" or "organization." In this artificial organization, every software agent has one or more roles, and can interact in order to achieve a common goal.

The MAS Framework is being used in many projects in the TecComm group, and in all of these projects we were able to reduce the implementation time of agent-based systems. The TSpace [10] server also enables our agents to cooperate in a distributed system over a network. We decided to implement the framework using Sun's Java [17], because we wanted to enable our software agents to run in almost any operating system that has a Java Virtual Machine. Consequently, the framework enables the construction of a very good agent-based system to run over the Internet.

References

[1] Weiss, G. *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press, Second printing, 2000.

[2] Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Pub Co, 1999.

[3] Garcia, A.; Silva, V.; Lucena, C.; Milidiú, R. *An Aspect-Based Approach for Developing Multi-Agent Object-Oriented Systems*. Simpósio Brasileiro de Engenharia de Software, Rio de Janeiro, Brasil, Outubro 2001.

[4] Garcia, A.; Lucena, C. J.; Cowan, D.D. *Engineering Multi-Agent Object-Oriented Software with Aspect-Oriented Programming*. Submitted to Practice & Experience, Elsevier, May 2001.

[5] Garcia, A.; Lucena, C. J. *An Aspect-Based Object-Oriented Model for Multi-Agent Systems*. 2nd Advanced Separation of Concerns Workshop at ICSE'2001, May 2001.

[6] M. Fayad, D. Schmidt. *Building Application Frameworks: Object-Oriented Foundations of Design*. First Edition, John Wiley & Sons, 1999.

[7] Wooldridge, M; Jennings, N. R.; Kinny, D. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Kluwer Academic Publishers.

[8] Kendall, E.; Krishna, P.; Pathak, C.; Suresh, C. *A Framework for Agent Systems*. In: Implementing Application Frameworks – Object-Oriented Frameworks at Work, M. Fayad et al. (editors), John Wiley & Sons, 1999.

[9] Silva, V.T.; Lucena, C.J.P. *Um Modelo Orientado a Objetos para Sistemas Multi-Agentes*. MCC30/01. Departamento de Informática. PUC-Rio. October 2001.

[10] IBM TSpaces Web Site - http://www.almaden.ibm.com/cs/TSpaces/

[11] Silva, O; Garcia, A; Lucena, C.J. T-Rex: *A Reflective Tuple Space Environment for Dependable Mobile Agent Systems*. III WCSF at IEEE MWCN 2001, Recife, Brasil, August 2001.

[12] Sardinha, J. A. R. P. *VGroups – Um framework para grupos virtuais de consumo*. Master's dissertation – Departamento de Informática – PUC-Rio. Mar/2001.

[13] Milidiu, R.L.; Lucena, C.J.; Sardinha, J.A.R.P. *An object-oriented framework for creating offerings*. 2001 International Conference on Internet Computing (IC'2001) June 2001.

[14] Flavio Bevilacqua, J. A. R. P. Sardinha. *Estruturas dinâmicas de incentivos para grupos de consumo*. Multi-agent system workshop. Departamento de Informática. PUC-Rio. July 2001.

[15] Ribeiro, P.C. *Modelagem e Implementação OO de Sistemas Multi-Agentes*. Master's Dissertations, Departamento de Informática, PUC-Rio, 2001.

[16] Fontoura, M.F.; Haeusler, E.H.; Lucena, C.J.P. *The Hot-Spot Relationship in OO Framework Design*. MCC33/98, Computer Science Department, PUC-Rio, 1998.

[17] Java Web Site - http://java.sun.com/