Preety, Nidhi Jindal, Vikas Tomar / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 2, March -April 2013, pp.877-882

# Optimizing the agents coordination in Multi-agent system through JADE Tool

**Preety*, Nidhi Jindal**, Vikas Tomar*****

*(Research scholar, Dr. K.N. Modi University, Newai, Rajasthan (India)
** (Assistant Professor, Dronacharya College of Engineering, Gurgaon, Haryana (India)
*** (Assistant Professor, Terna Engineering College, Nerul, Navi Mumbai (India)

## ABSTRACT

**Agent-based systems technology has generated a lot of enthusiasm in recent years because of its guarantee as a innovative standard for conceptualizing, designing, and implementing software systems. Increasingly, however, applications require multiple agents that can work together. A multi-agent system (MAS) is a loosely coupled network of software agents that interact to solve problems that are beyond the individual capacities or knowledge of each problem solver.**

**JADE (Java Agent Development Framework) is a software environment to build agent systems for the management of networked information resources. JADE offers an agent middleware to implement efficient FIPA2000 compliant multi-agent systems and supports their development through the availability of a predefined programmable agent model, an ontology development support, and a set of management and testing tools. In this paper we will use JADE tool for developing the autonomous software agents which manage (intermediates) the communication and coordination between an agent and the agent society wherein this is situated. With this aim, we have used the Java agent development toolkit provides agents with a highly versatile range of programmable before and during the agent's run-time communication and coordination services.**

**Keywords– Intelligent Agent, Multi-agent system, Reasoning, Coordination, JADE**

## I. INTRODUCTION

Multi-Agent systems or Agent-based Systems: provide a way of conceptualizing sophisticated software applications that face problems involving multiple and (logically and often spatially) distributed sources of knowledge.
It can be thought of as computational systems composed of several agents that interact with one another to solve complex tasks beyond the capabilities of an individual agent.

From the agent perspective, completely optimal agents are not really practicable. Agents are faced with all sorts of limitations. Some limitations may physically prevent certain behavior, e.g., a soccer robot with acceleration constraints.
Other limitations are self-imposed to help guide an agent's learning, e.g., using a subproblem solution for advancing the ball down the field. In short, limitations prevent agents from playing optimally and possibly from following a Nash equilibrium. This clash between the concept of equilibria and the reality of limited agents is a topic of critical importance. Do equilibria exist when agents have limitations? Are there classes of domains or classes of limitations where equilibria are guaranteed to exist? One method for deciding what strategy to use when negotiating with a particular agent is to use a model based approach that tries to construct a model of the agent and then based on this model select a strategy. There are a number of reasons why this approach would be difficult to use in practice. Firstly, obtaining an accurate enough model of another agent is a very difficult learning problem, since the only interaction agents have is through the exchange of times when they negotiate meetings. From this information, it is hard to make accurate conclusions about what times an agent prefers, how busy the agent is, what negotiation strategy it is employing etc. Secondly, to build a model of another agent, many training examples are required. It would be preferable if an agent was able to learn to negotiate, while actually negotiating.

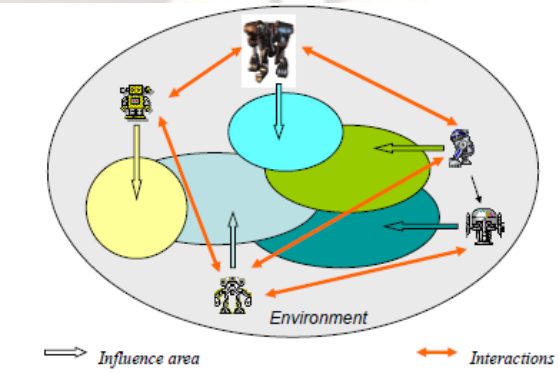


Figure 1: Multi-agent system

Goal, plan, and policy hierarchies have proven to be very successful methods for coordinating agents. In these approaches we assume the existance of one of these hierarchies and the

problem then becomes that of determining which parts of the hierarchy are to be done by which agents. In this setting we assume that the agents are coopertive, that is, they will do exactly what we tell them to. The problem is one of finding an good-enough answer

## II. ADVANTAGES OF MULTI-AGENT SYSTEM

An MAS has the following advantages over a single agent or centralized approach:

- An MAS distributes computational resources and capabilities across a network of interconnected agents. Whereas a centralized system may be plagued by resource limitations, performance bottlenecks, or critical failures, an MAS is decentralized and thus does not suffer from the "single point of failure" problem associated with centralized systems.
- An MAS allows for the interconnection and interoperation of multiple existing legacy systems. By building an agent wrapper around such systems, they can be inporporated into an agent society.
- An MAS models problems in terms of autonomous interacting component-agents, which is proving to be a more natural way of representing task allocation, team planning, user preferences, open environments, and so on.
- An MAS efficiently retrieves, filters, and globally coordinates information from sources that are spatially distributed.
- An MAS provides solutions in situations where expertise is spatially and temporally distributed.

An MAS enhances overall system performance, specifically along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility, and reuse. We are currently interested in interagent communication and coordination, and are building reusable multi-agent applications that facilitate interaction among different kinds of agent systems

## III. RELATED WORK

Thomas Wagner and Victor Lesser et al. (1991) proposed *MQ* or *motivational quantities* framework addressed the issues by evaluating candidate tasks based on the agent's organizational context and by framing control as a local agent optimization problem that approximates the global problem through the use of state and preference. Jaime Simao Sichman et al. (1995) presented briefly both our agent model and the theory of dependence on which our social reasoning mechanism is based, and illustrate how such an adaptation can be achieved using a very simple example of the blocks world.

Onn Shehory et al. (1998) demonstrated multiagent systems developed to date have several common architectural characteristics, even though differences in their design and implementation result in variations in their strengths and weaknesses. A large portion of the research in the design and implementation of MAS addresses questions such as: given a computational problem—can one build a MAS to solve it? What should be the properties of this MAS given the problem? Having developed a MAS, what is the class of problems that this MAS, either as developed or with slight modifications, can solve?

Peter Stone et al. (2000) performed the survey of MAS is intended to serve as an introduction to the field and as an organizational framework. A series of general multiagent scenarios are presented. For each scenario, the issues that arise are described along with a sampling of the techniques that exist to deal with them. The presented techniques are not exhaustive, but they highlight how multiagent systems can be and have been used to build complex systems. When options exist, the techniques presented are biased towards machine learning approaches. Additional opportunities for applying machine learning to MAS are highlighted and robotic soccer is presented as an appropriate test bed for MAS. This survey does not focus exclusively on robotic systems. However, we believe that much of the prior research in non-robotic. MAS is relevant to robotic MAS, and we explicitly discuss several robotic MAS, including all of those presented in this issue. Martin L Griss, Steven Fonseca et al. (2002) implemented SmartAgent extends JADE behaviors with uniform message and system events, a multi-level tree of dispatchers that match and route events, and a hierarchical state machine that is based on the UML state chart model. Adherence to the UML helps bridge the object-oriented to the agent-oriented programming using an industry familiar modeling language and tools. Combining events, dispatcher tree and hierarchical state machines simplifies programming of default and context dependent behavior. This hypothesis was confirmed in a meeting scheduler prototype where code previously written using pure JADE was refactored using SmartAgent. Rajveer Basra et al. (2005) reported on an investigation in to how a Multi-Agent System (MAS) may be used for resolving scheduling issues for LU. It is a previously unexplored domain. A prototype system MASLU is developed through the use of Multi-agent Systems (MAS) technology, in an innovative and unique manner, with a view to resolving the London Undergrounds scheduling/logistics issues in real time.

Kalliopi Kravari et al. (2010) reported on the implementation of EMERALD, a knowledge-

based framework for interoperating intelligent agents in the Semantic Web. More specifically, a multi-agent system was developed on top of JADE, featuring trusted, third party reasoning services, a reusable agent prototype for knowledge-customizable agent behavior, as well as a reputation mechanism for ensuring trust in the framework. Finally, a use case scenario was presented that illustrates the viability of the proposed framework.

Ghulam Mahdi et al. (2010) presented an integrated view of coordination covering agent reasoning, message passing, resource management and negotiation. We argued for an integrated and comprehensive approach of real-time coordination" in one unified model of coordination. Our position regarding real-time agent coordination would result in overall better understanding of real-time coordination and performance amelioration in MASs. We analyze current approaches and present an outline for integrated and comprehensive view of\real time coordination".

## IV. JADE TOOL

JADE (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. It includes two main products: a FIPA-compliant agent platform and a package to develop Java agents. JADE has been fully coded in Java. JADE is written in Java language and is made of various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks. Java is the programming language of choice because of its many attractive features, particularly geared towards object-oriented programming in distributed heterogeneous environments; some of these features are Object Serialization, Reflection API and Remote Method Invocation (RMI).

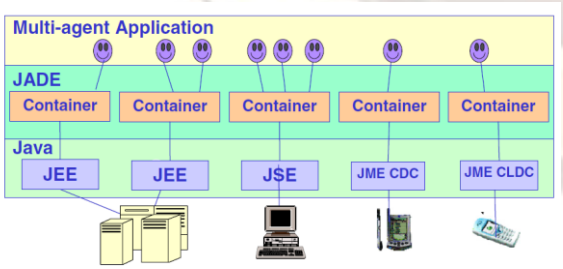


Figure 2: Architecture of JADE

JADE is composed of the following main packages.

- jade.core implements the kernel of the system. It includes the Agent class that must be extended by application programmers.
- jade.core.behaviours sub-package contains Behaviour class hierarchy. Behaviours implement the tasks, or intentions, of an agent. They are logical activity units that can be composed in various ways to achieve complex execution patterns and that can be concurrently executed. Application programmers define agent operations writing behaviours and agent execution paths interconnecting them.
- jade.lang.acl sub-package is provided to process Agent Communication Language according to FIPA standard specifications.
- jade.content package contains a set of classes to support user-defined ontologies and content-languages. A separate tutorial describes how to use the JADE support to message content. In particular jade.content.lang.sl contains the SL codec, both the parser and the encoder.
- jade.domain package contains all those Java classes that represent the Agent Management entities defined by the FIPA standard, in particular the AMS and DF agents, that provide life-cycle, white and yellow page services. The subpackage jade.domain.FIPAAgentManagement contains the FIPA-Agent-Management Ontology and all the classes representing its concepts. The subpackage jade.domain.JADEAgentManagement contains, instead, the JADE extensions for AgentManagement (e.g. for sniffing messages, controlling the life-cycle of agents), including the Ontology and all the classes representing its concepts. The subpackage jade.domain.introspection contains the concepts used for the domain of discourse between the JADE tools (e.g. the Sniffer and the Introspector) and the JADE kernel. The subpackage jade.domain.mobility contains all concepts used to communicate about mobility.

jade.gui package contains a set of generic classes useful to create GUIs to display and edit Agent-Identifiers, Agent Descriptions, ACLMessages. jade.mtp package contains a Java interface that every Message Transport Protocol should implement in order to be readily integrated with the JADE framework, and the implementation of a set of these protocols. JADE can be run by 'java jade.Boot –gui'.

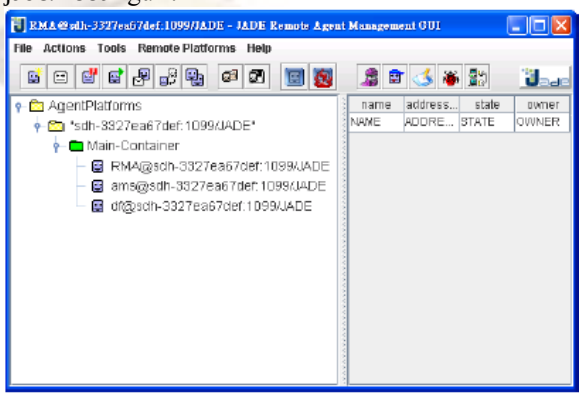


Figure 2: JADE screen

## V. ORGANIZATIONS & ROLES IN JADE TOOL

We introduce organizations and roles as first class entities in JADE, with behaviours, albeit not autonomously executed, and communication abilities. Thus, organizations and roles can be implemented using the same primitives of agents by extending the JADE Agent class with the classes Organization and Role. Analogously, to implement autonomous agents who are able to play roles, the Player class is defined as an extension of the Agent class. The Role class and its extensions represent the role types. Their instances represent the role instances associated with an instance of the Agent.
Organizations and roles, however, differ in two ontological aspects: first roles are associated to players; second, roles are not independent from the organization offering them. Thus, the Role class is subject to an invariant, stating that it can be instantiated only when an instance of the organization offering the role is present. Conversely, when an organization is destroyed all its roles must be destroyed too.

A further difference of role classes is that to define "powers", they must access the state of the organization they belong too. To avoid making the state of the organization public, the standard solution offered by Java is to use the so-called "inner classes".

Inner classes are classes defined inside other classes ("outer classes"). An inner class shares the namespace of the outer class and of the other inner classes, thus being able to access private variables and methods. The class Role is defined as an inner class of the Organization class. Class extending the Role class must be inner classes of the class extending the Organization class. In this way the role can access the private state of the organization and of the other roles. Since roles are implemented as inner classes, a role instance must be on the same platform as the organization instance it belongs to. Moreover, the role agent can be seen as an object from the point of view of the organization and of the other roles which can have a reference to it, besides sending messages to it. In contrast, outside an organization the role agent is accessed by its player (which can be on a different platform) only as an agent via messages, and no reference to it is possible. So not even its public methods can be invoked.

## VI. IMPLEMENTING ORGANIZATION OF MAS IN THROUGH JAVA INTEREGENTS

To implement an organization it's necessary to extend Organization, subclass of Agent, which offers protocols necessary to communicate with agents who want to play a role, and the behaviours to manage the information about roles and their players. Moreover, the Organization class includes the definition of the Role inner class that can be extended to implement new role classes in specific organizations. To support the creation and management of roles the Organization class is endowed with the (private) data structures and (private) methods to create new role instances and to keep the list of the AIDs (Agent IDs) of role instances which have been created, associated
with the AIDs of their players. Since roles are Java inner classes of an organization, the organization code can be written in Java mostly is regarding what is a JADE application.

Moreover, the inner class mechanism allows the programmer to access the role state and viceversa, while maintaining the modularity character of classes. The organization listens from messages from any agent (even if some restrictions can be posed at the moment of accepting to create the role), while the subsequent communication between player and role is private. After a request from an agent the behavior representing the protocol forks creating another instance of itself to be ready to receive requests of other agents in parallel.

The first message is sent by the player as initiator and is a request to enact a role. The organization, if it considers the agent authorized to play the role, returns to the candidate player a list of specifications about the powers and requirements of the requested role which are contained in its knowledge base, sending an inform message containing the list; otherwise, it denies to the player to play the role, answering with an inform message, indicating the failure of the procedure. In case of positive answer,
the player, invoking the method canPlay using the information contained in the player about the requirements, decides whether to respond to the organization that it can play the role (agree) or not (failure).

All Responder behaviours, instead, are cyclic and they are rescheduled as soon as they reach any final state of the interaction protocol. Notice that this feature allows the programmer to limit the maximum number of responder behaviours that the agent should execute in parallel. For instance, the following code ensures that a maximum of two contract-net tasks will be executed simultaneously.

```
Protected void setup() {
addBehaviour(new
FipaContractNetResponderBehaviour(<arguments
>));
addBehaviour(new
FipaContractNetResponderBehaviour(<arguments
>));
}
```

It is intention of the programmer to keep only this couple of classes and soon deprecate the other jade.proto classes. It has also ContractNetInitiator/Responder have been implemented that offer API and functionalities with

the same style and that replace the deprecated old FipaContractNetInitiator / ResponderBehaviour.

## VII. CONCLUSION

In this paper we use the ontological model of organizations proposed in organizations. We use as agent framework JADE since it provides the primitives to program MAS in Java. We define a set of Java classes which extends the agent classes of the JADE to have further primitives for building organizations structured into roles. To define the organizational primitives JADE offered advantages but also posed some difficulties. First of all, being based on Java, it allowed to reapply the methodology used to implement roles in powerJava to implement roles as inner classes. Moreover, it provides a general purpose language to create new organizations and roles. Finally, being based on FIPA speech acts, it allows agents programmed in other languages to play roles in organizations, and viceversa, JADE agents to play roles in organizations not implemented in JADE. However, the decision of using JADE has some drawbacks. For example, the messages used in the newly defined protocols can be intercepted by other behaviours of the agents. This shows that a more careful implementation should use a more complex communication infrastructure to avoid this problem. Moreover, since JADE behaviours differently from methods do not have a proper return value, they make it difficult to define requirements and powers. Finally, due to the possible parallelism of behaviours inside an agent, possible synchronization problems can occur.

## REFERENCES

[1]. M. Allen-Williams. Coordination in multi-agent systems. PhD thesis, University of Southampton, 2006.

[2]. M. Bouroche, B. Hughes, and V. Cahill. Real-time coordination of autonomous vehicles. In *IEEE Conference on Intelligent Transportation Systems* (ITSC 06), 2006, 1232-1239.

[3]. C. Carrascosa, J. Bajo, V. Julian, J. Corchado, and V. Botti. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems with Applications*, 34(1), 2008, 2-17.

[4]. P. Ciancarini, A. Omicini, and F. Zambonelli. Multiagent system engineering: The coordination viewpoint. *Lecture notes in computer science, pages*, 2000, 250-259.

[5]. E. Durfee. Scaling up agent coordination strategies. *Computer*, 2001, 39-46.

[6]. E. Durfee. Challenges to Scaling-Up Agent Coordination Strategies. *Multiagent Systems, Artificial societies and simulated organizations*, 10, 2004, 119-132.

[7]. D. Gelernter and N. Carriero. Coordination languages and their significance. *Communication ACM*, 35(2), 1992, 97-107.

[8]. N. Jamali and S. Ren. A layered architecture for real-time distributed multi-agent systems. *SIGSOFT Softw. Eng. Notes*, 30(4), 2005, 1-8.

[9]. E. D. Jensen. Real-time for the real world. http://www.real-time.org/

[10]. V. Julian and V. Botti. Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering*, 11(2), 2004, 135-149.

[11]. V. Julian, J. Soler, M. Moncho, and V. Botti .Real-Time Multi-Agent System Development and Implementation. *Recent Advances in Artificial Intelligence Research and Development*, page 333, 2004.

[12]. J. Kim, H. Shim, H. Kim, M. Jung, I. Choi, and J. Kim. A cooperative multi-agent system and its real time application to robot soccer. In *IEEE International Conference on Robotics and Automation*, IEEE, 1997, 638-647.

[13]. S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. *Artificial Intelligence*, 75(2), 1995, 297-345.

[14]. X. Liu, X. Zhang, L. Soh, J. Al-Jaroodi, and H. Jiang. A distributed, multiagent infrastructure for real-time, virtual classrooms. In Proceedings of the *International Conference on Computers in Education (ICCE2003)*, Hong Kong, 2003, 2-5.

[15]. C. Micacchi and R. Cohen. A framework for simulating real-time multi-agent systems. *Knowledge and Information Systems*, 17(2), 2008, 135-166.

[16]. M. Mock and E. Nett. Real-time communication in autonomous robot systems. In Proceedings of the *The Fourth International Symposium on Autonomous Decentralized Systems*, page 34, USA, 1999.

[17]. H. Nwana, L. Lee, and N. Jennings. Co-ordination in multi-agent systems. *Lecture Notes in Computer Science*, 1198, 1997, 42-58.

[18]. C. Sierra and L. Sonenberg. A real-time negotiation model and a multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, 11(1), 2005, 5-6.

[19]. L. Soh and C. Tsatsoulis. A real-time negotiation model and a multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, 11(3), 2005, 217-271.

[20]. J. Soler, V. Julian, M. Rebollo, C. Carrascosa, and V. Botti. Towards a real-time multi-agent system architecture. *COAS, AAMAS*, 2002.

[21]. J. Stankovic, T. Abdelzaher, C. Lu, L. Sha, and J. Hou. Real-time communication and coordination in embedded sensor networks. *Proceedings of the IEEE*, 91(7), 2003, 1002-1022.

[22]. M. Baldoni, G. Boella, and L. van der Torre. Modelling the interaction between objects: Roles as affordances. In Proc. of KSEM 2006, LNCS 4092, pp. 42–54. Springer, 2006.

[23]. M. Baldoni, G. Boella, and L. van der Torre. Interaction between Objects in powerJava. Journal of Object Technology, 6(2):7–12, 2007.

[24]. F. L. Bellifemine, G. Caire, and D. Greenwood. Developing Multi-Agent Systems with JADE. Wiley, 2007.

[25]. G. Boella, R. Damiano, J. Hulstijn, and L. van der Torre. ACL semantics between social commitments and mental attitudes. In Proc. of AC 2005 and AC 2006, LNAI 3859, pp. 30– 44. Springer, 2006.

[26]. G. Boella, V. Genovese, R. Grenna, and L. der Torre. Roles in coordination and in agent deliberation: A merger of concepts. In Proc. of Multi-Agent Logics, PRIMA 2007.

[27]. G. Boella and L. van der Torre. Organizations as socially constructed agents in the agent oriented paradigm. In Proc. of ESAW'04, LNAI 3451, pp. 1–13, 2005. Springer.