Comparing Travelling Design Patterns for Mobile Agent Development Using JADE

Nikolaos Karagiannis
Dept. of Informatics
Technological Educational Institute (T.E.I.) of Athens
Athens, Greece
ms09056@teiath.gr

Konstantinos Giannakos
Dept. of Informatics
University of Piraeus
Piraeus, Greece
kon.giannakos@gmail.com

Konstantinos Antonis
Dept. of Informatics and Computer Technology
Technological Educational Institute (T.E.I.) of Lamia
Lamia, Greece
k_antonis@teilam.gr

***Abstract* - Mobile agents are autonomous processes that are used to assign various tasks. Those processes are migrating to several nodes to execute those tasks locally, instead of RPCs. Their migration way may be different according to the application type and it is based on a design pattern. Here, we present comparative results of three different travelling design patterns for mobile agents (Itinerary, Branching and Star-shaped) with the use of an application that we developed. Derived results showed that the branching pattern performs better than the other two in terms of turnaround times, whether we use constant size or variable size of answers to mobile agent requests to servers.**

Keywords - Mobile Agents; Travelling Design Patterns; JADE Application

I. INTRODUCTION

An agent is a special software component providing an interoperable interface to an arbitrary system and/or behaving like a human agent working for some clients in pursuit of its own agenda. Some common tasks for such software components are monitoring of systems, searching for specific information, managing a system, etc. Agent – based methods are becoming more and more popular as time goes by and they are used in many different fields (like economics, e.g., [22]). Some common characteristics of agents are autonomy, sociality, intelligence and mobility [5].

Agents can have a combination of various characteristics. Mobile agents (having as basic characteristic the mobility) are able to migrate from one computer to another autonomously and continue its execution on the destination computer. They are used instead of RPCs (Remote Procedure Calls), exchanging remotely various data. The most basic advantage of mobile agents is the reduction of the used bandwidth, because the agent migrates itself and there is no data exchange between different procedures hosted on different computers (Figure 1) [5].

Also, these procedures are asynchronous and autonomous, so their function depends on network connectivity. That means that there is no need for continuous connection between nodes for data exchange. In the RPC model, if the connection stops when the processes are exchanging data, then the connection has to be restarted. In the meantime, a process waits for a response from the other process aimlessly. But, with the use of a mobile agent a job can be completed locally while the connection is down, and finally waits to migrate until the connection is established again (Figure 2). This is also useful in mobile phones in cases of unstable connections [5].

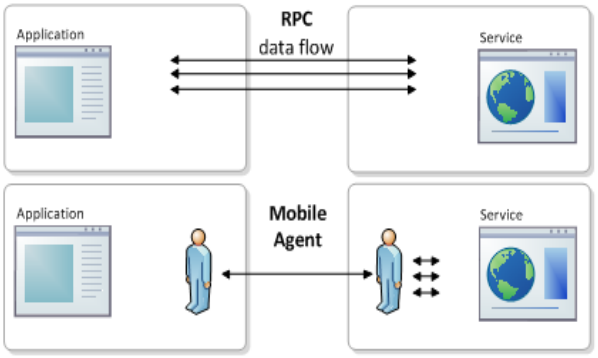

Figure 1. Agent migration vs data migration in RPC.

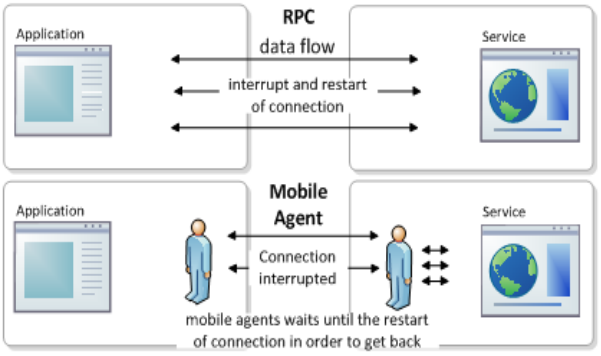

Figure 2. There is no need of continious network connection.

But also, it must also be considered that there are several difficulties/disadvantages of the mobile agent technology application into the internet infrastructures. The most important difficulties are various security problems, the high computation cost required by a server in order to host and serve a lot of mobile agents concurrently and the high difficulty of the creation and the application of those infrastructures.

Multi Agent Systems (MAS) are systems in which many agents interact in order to solve a common problem. The problem is divided into several sub problems distributing each one to different agents in the MAS system. A MAS system works on a set of various computers connected via a specific network (LAN or WAN, etc). A MAS system is very appealing for building open and distributed applications [13]. Various MAS systems can communicate in order to achieve several user needs [5].

It should also be stated that software agents, which bring together the two concepts "process" and "object", are interesting building blocks for flexible system architectures, even if they are not always mobile. On the one hand, mobile agents provide a novel and useful example for an open and distributed MAS ([14], [15], [16]). On the other hand, static agents (non-mobile) are probably as important as mobile agents: they encapsulate autonomous activities in a stronger way than classical objects, they communicate with other (mobile) agents via the same protocols and interfaces, and they provide (with mobile agents) a uniform way to structure large distributed systems [2].

A variety of design patterns for mobile agents have been proposed in the past organized in different categories [6]. A basic task pattern is the Master-Slave pattern. On this pattern, a master agent delegates a task to be done on a given agency to a slave agent(s). The slave agent visits the indicated agency where it accomplishes the task, and then returns to the source agency carrying the results. The master agent receives the results and then the slave destroys itself. The migration procedure of an agent varies also, creating the category of travelling design patterns [6]. The itinerary pattern (Figure 3) provides a way to execute the migration of an agent, which will be responsible for executing a given task in remote hosts. The agent receives an itinerary on the source agency, indicating the sequence of agencies it should visit. Once in an agency, the agent executes its task locally and then continues on its itinerary. After visiting the last agency, the agent goes back to its source agency [1].

On the branching travelling pattern (Figure 4), the agent receives a list of agencies to visit and clones itself according to the number of agencies in the itinerary. Then, all clones will visit an agency of the received list. Each clone has to execute its corresponding task and notify the source agency when the task is completed. The importance of this pattern is that it splits the tasks that can be executed in parallel [1]. A typical example would be a search agent that sends out slave agents to visit multiple machines in parallel. Of course, mechanisms to control the high degree of dynamism of such agent-enabled parallel computations then become a necessity [2].

So, we can imagine the World Wide Web consisting of servers and clients working on mobile agent platforms exchanging mobile agents. Also, by integrating extra characteristics like intelligence and sociality we will have smarter applications offering high level services (e.g., auto-learning) and achieving better and more specific results.

On the Star-Shaped travelling pattern (Figure 5) the agent receives a list of agencies that it has to visit. So, it migrates to the first destination agency, where it executes a task, going back to the source agency. The agent repeats this cycle until visiting the last agency on its list [1].

In this paper we compare the three above mentioned design patterns with the use of a mobile MAS application that we have implemented. The application contains two static agents representing a web client and a web server (hosted on different machines). Both client and server exchange mobile agents. We developed all those agents using the JADE platform and we execute them in many nodes. A lot of implementations for different design patterns have been proposed in the past (e.g., [1], [6], [8]), but it is not our purpose to present another alternative implementation on the same subject. The contribution of our work is the presentation of comparative results for those patterns in terms of turnaround times, for constant and variable sizes of answers to mobile agent requests from the implemented servers.

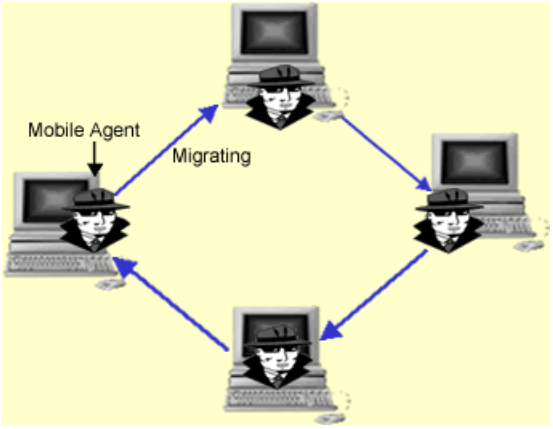

Figure 3. Itinerary pattern.

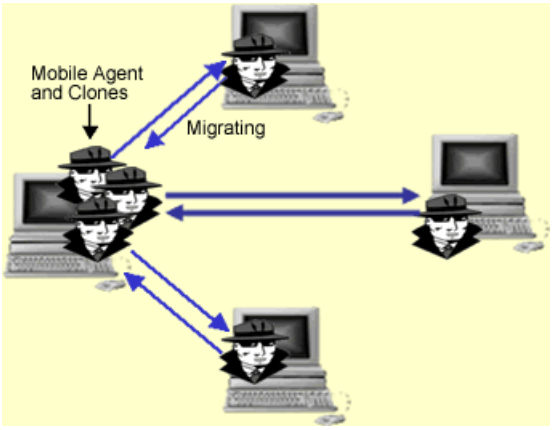

Figure 4. Branching pattern.

 ISBN: 978-1-61208-202-8

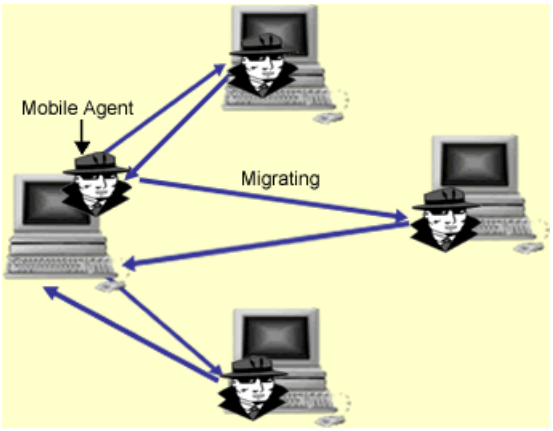

Figure 5. Star-shaped pattern.

II. RELATED WORK

Aridor and Lange [6] reported on several design patterns for mobile agencies classified in three different categories: travelling, task and interaction patterns. Travelling patterns encapsulate mobility management of an agent for one or more destinations. Task patterns are concerned with the breakdown of a task and how these tasks are delegated to one or more agents. Interaction patterns are concerned with locating agents and facilitating their interactions. They also implemented three of them (master-slave, meeting and itinerary) and shared their experiences with it. Eight different agent design patterns are implemented in [1] in JADE: Itinerary, Star-Shaped, Branching, Master-Slave, MoProxy, Meeting, Facilitator, and Mutual Itinerary Recording. The itinerary, branching and star-shaped patterns were proposed in [11].

Kendall et al. [17] present several patterns of intelligent and mobile agents based on a layered architecture considering mobility and intelligence separately. A set of seven patterns related to agent communication mechanisms are discussed in [18], but they do not take into account the intelligence and mobility together.

Eshtay [7] proposed the Hierarchal Traveling design pattern, which is a combination of the itinerary pattern and the depth first algorithm. This pattern was implemented in JADE, too. Wang et al. [9] uses the branching pattern to develop in JADE an agile supply chain management model.

Ojha et al. [12] propose a design pattern for intelligent mobile agents. It helps in efficient mobility of these agents, which are more often fatty. It enables dynamic on-demand behaviour specific to a network host environment. It describes the pattern using a suitable pattern template and reported the results of its implementation (using JADE) in a prototype multi-agent system for e-commerce domain.

Maamar and Labbe [10] described two strategies (servlet and applet) that could enhance the operations of software agents and showed that both strategies could suit them. Agents should be embedded with mechanisms that allow them to make the correct decision: either move or invite.

There are also a lot of design patterns proposed in the past, that they do not consider mobility of agents, but they are based on social and intentional characteristics of an agent (e.g., [19],[20], [21]).

III. OUR APPLICATION: A WEB AGENT EXPLORER

We developed a web application that informs the client about the latest registrations that are added to various e-news sites that interests him. Several mobile agents (each one representing a user) are migrating to various servers to retrieve information that the user is interested in.

The application developed with the use of the JADE platform. It was developed to help us compare the itinerary, star-shaped and branching design patterns under various circumstances and extract useful conclusions. We chose to compare only these three patterns because, one the one hand, they are of the most famous patterns in the research community. On the other hand, our application is too simple and does not involve collaboration or interactions between agents and does not check permissions. So, patterns like the Meeting, or MoProxy, etc, are not suitable for it. The application obeys the master – slave model containing two static agents (masters) representing a web client and a web server. Both client and server exchange mobile agents (slaves) obeying the three above mentioned design patterns, alternatively and we execute them in many nodes. We mention here, that our system is still in prototype level.

IV. The JADE Platform/Framework

JADE (Java Agent Development Framework) is a platform supporting agent processes and also offers libraries (framework) for multi agent application development written in JAVA. It is ideal for distributed application development based on multi agent systems. For application development, JADE has an IDE with useful tools and a GUI for platform administration. The platform offers all the necessary services to the agents that they are installed on it. With some of those services, agents can identify and communicate each other, and they can search each other after they have registered on specific platform catalogues [3].

Each platform constitutes a MAS with at least one container. Each container is installed to a computer and it can support agents and offer them all the necessary services. Consequently, a platform can constitute a network of containers (e.g., a LAN) (Figure 6). Two basic services are the AMS (Agent Management System) and DF (Directory Facilitator) directories that are local agents [3], [4].

Agents are java classes inheriting the Agent class of JADE libraries. The actual job, or jobs, an agent has to do is carried out within ‘behaviours’. A behaviour represents a task that an agent can carry out. An agent can execute several behaviours concurrently. The scheduling of behaviours in an agent is not pre-emptive (as for Java threads), but cooperative. This means that when a behaviour is scheduled for execution its method is called and runs until it returns.

 ISBN: 978-1-61208-202-8

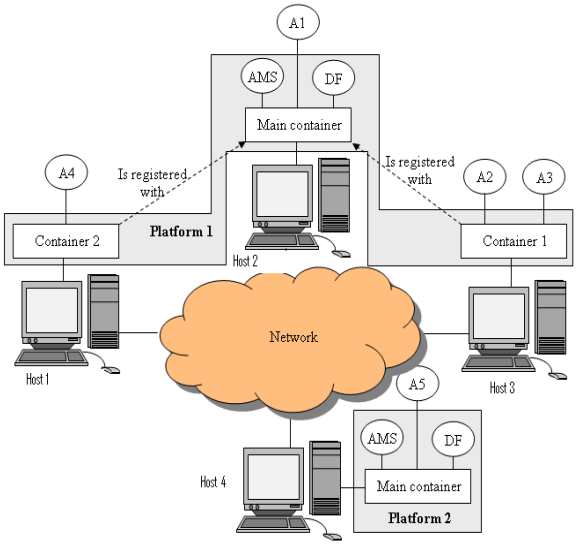

Figure 6. JADE architecture overview.

Therefore, it is the programmer who defines when an agent switches from the execution of a behaviour to the execution of another [3]. Here, we use three different behaviour types:

- OneShotBehaviour: executes once and dies.
- CyclicBehaviour: stays active as long as its agent is alive and is called repeatedly after every event.
- TickerBehaviour: periodically executes some user-defined piece of code.

V. *Functional Description*

Each user interacts with a local agent via a GUI. User enters to GUI the web domains he is interested in. Consequently, the local agent sends mobile agent(s) according to the mobility pattern that has been set. When the above mobile agent(s) returns having the available pages of each domain, the user is able to select the pages he wants to retrieve the latest updates of them. Next, the user declares the frequency for a mobile agent(s) to visit the selected domains in order to check if the selected pages have been updated. For instance, a user may want the mobile agent(s) to migrate to check those pages every 5 minutes. For the first time, a mobile agent returns the latest registrations from those pages, and only when there have been updates since his latest visit for all the other times. When a mobile agent arrives at a server, it is served by a local agent that "lives" there.

VI. *Architecture*

Client and Server processes are hosted in different JADE platforms and each one constitutes a different MAS (JADE platforms). Until now, JADE does not support agent mobility between containers that belong to different platforms (inter platform mobility). We used the IPMS addon that provides this feature [23]. The client corresponds to one container (computer) and the server may be distributed to many containers, but in our approach it is constituted by one container.

1) *Client Components Description:* The client side container hosts a local agent except the basic JADE agents (like AMS) (Figure 7). When the application starts, AMS creates an object of this agent. This local agent is static and provides the user a GUI to enter his preferences and receive the results. It also checks and manages the information that mobile agents return. So, according to the Master-Slave pattern, the local agent forwards tasks to a mobile agent(s). It is able to save data to disk, while a mobile agent is not. The mobile agent sends data to a local agent. We created those mechanisms for system security reasons. The static agent has to be capable to authenticate the returned mobile agent preventing from malware attacks. The local agent creates different types of mobile agents classes for each migration pattern. When a mobile agent returns back to the client host, it sends the data collected during migration to the local agent and then destroys itself.

2) *Server components description (Figure 8):* The server side container hosts a local agent too, except the basic JADE agents (like AMS and DF). When the application starts, AMS creates an object of this agent. The local agent publishes in DF the services provided. Those services correspond to the pages that mobile agents can retrieve updates. The local agent is static and serves the incoming mobile agents. The arrived agents send requests to the local server about the info a user is interested. So, the role of this agent is to provide an interface between the arriving agents and the database and to protect the system from malware attacks.

VII. EXPERIMENTAL RESULTS

We compared the three travelling patterns which were mentioned before using our application. We set up one client and three servers into a LAN network. The client was set up on a four core 64bit CPU at 3.3 GHZ with 8GB RAM. Servers were created as 4 virtual machines. One of the servers was set up in a virtual machine of the client's machine having 512 MB RAM, while the others were set up on another machine containing a CPU at 2 GHZ with 512MB RAM.

Throughout the paper we use the term task to refer to the trip of a mobile agent to the 4 servers mentioned above in order to collect data. Specifically, the client agent keeps a hash table, where each key corresponds to the name of a site the user is interested (e.g., e-news.com, games.com, e-shop.gr). Each key indexes a data structure containing the specific pages inside a site, as well as the date and time of the last visit. All data exchanged between a server and a client, are encrypted with SSL. An instance of a mobile agent is created in the client side for each task, containing the appropriate records from this table. The server receives those records and registers the data collected into a message, along with other information like the structure of the page, formatting of the page, etc. Since the information exchanged is in text format, there is no significant overhead for the server to serve a mobile agent.

 ISBN: 978-1-61208-202-8

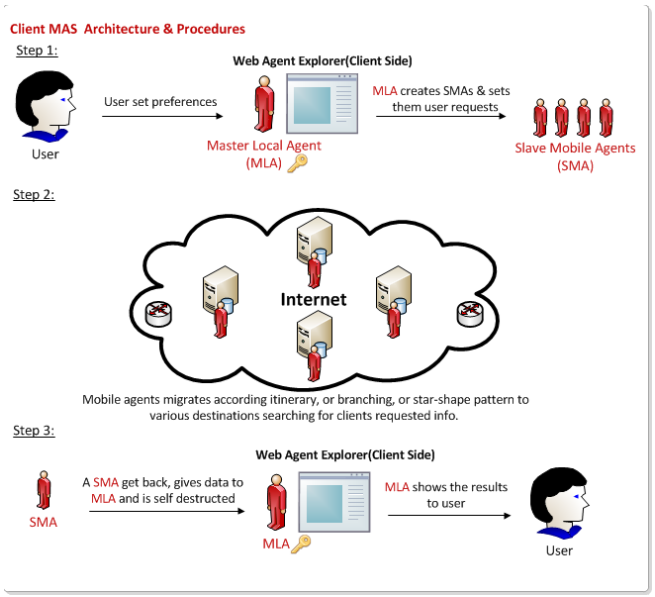

Figure 7. Client MAS Architecture.

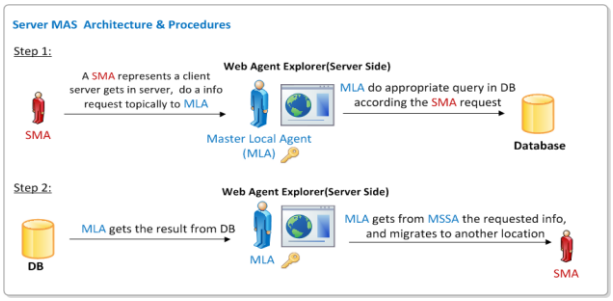

Figure 8. Server MAS Architecture.

Using itinerary and star-shaped patterns we have one mobile agent for each task. Each task starts when a mobile agent is created and ends after having visited all the nodes and returning to client sending the results to the Master Agent. The turnaround time of an agent depends on the workload of the server and the number of the exchanged messages. In the branching pattern case, a mobile agent is sent in parallel for each task. The turnaround time of the task ends when the last mobile agent returns back. The size of the answer to a query imposed by a Server Agent, corresponding to a mobile agent request, is small because their content is in text format. We consider constant and variable size of an answer to measure the turnaround time (in seconds) for each task for the three travelling patterns (itinerary, branching and star-shaped).

A. *Constant Size of Answers*

We consider 20 tasks arriving sequentially. Upon the arrival of a task, a mobile agent(s) migrates automatically, from the client to the above servers and retrieves 28 database records of the same size from each server. We run the application for each pattern separately.

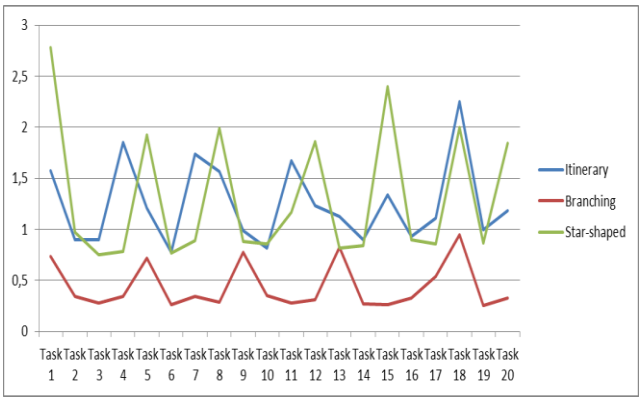

Figure 9. Comparing the three design patterns with constant size of answers.

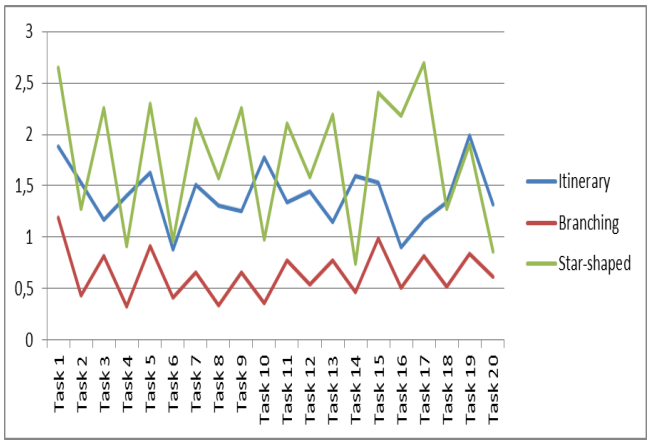

Figure 10. Comparing the three design patterns with variable size of answers.

Figure 9 presents the derived results of turnaround times per task for constant size of answers. Results show that the branching pattern achieves the best turnaround times for all tasks. The itinerary pattern performs a bit better than star-shaped on the average, but results are comparable. This is due to that some servers are hosted on the same machine as different virtual machines, although the star-shaped pattern sends more messages (48 messages vs 42, respectively). This performance difference would be greater if servers could be hosted in different machines.

B. *Variable Size of Answers*

We consider again 20 tasks arriving sequentially. But in this case we have different sizes of answers in each task. We developed a simple application (News Generator) in JAVA that runs in each server and adds some fake news updates (text format) to the database for various categories (like sports, weather news, politics, etc). The database update frequency varies, but it is less or equal to the task arrival time frequency in any case. We run again the application for each migration pattern separately and Figure 10 illustrates the derived results. Results show that the branching pattern achieves the best turnaround times for all tasks. The itinerary pattern performs better than star-shaped on the average.

 ISBN: 978-1-61208-202-8

VIII. CONCLUSIONS AND FUTURE WORK

A lot of design patterns for mobile agents have been proposed in the past. In this paper, we focus only on travelling design patterns and we try to compare three of the most commonly used of them: itinerary, branching and star-shaped, with the use of a mobile MAS application that we have implemented. The application contains two static agents representing a web client and a web server (hosted on different platforms). Both client and server exchange mobile agents. We developed all those agents using the JADE platform and we execute them in many nodes. The above mentioned design patterns are compared in terms of turnaround times, for constant and variable sizes of answers to mobile agent requests from the implemented servers. The derived results show very clearly that the branching pattern performs better under both circumstances.

In our future work, we intend to investigate hybrid implementations of design patterns and evaluate them on our platform. Derived results will be compared with the evaluation results of the current work and with other results assuming hybrid implementations.

REFERENCES

[1] Emerson Ferreira de Araújo Lima, Patrícia Duarte de Lima Machado, Jorge César Abrantes de Figueiredo, and Flávio Ronison Sampaio, "Implementing Mobile Agent Design Patterns in the JADE framework", EXP in Search of Innovation,Vol.-3, No.-3, September 2003.

[2] Stefan Funfrocken and Friedemann Mattern, "Mobile Agents as an Architectural Concept for Internet-based Distributed Applications The WASP approach", Proceedings of the KiVS'99, Springer-Verlag, pp. 32-43, 1999.

[3] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood, "Developing Multi Agent Systems with JADE", John Wiley & Sons, Ltd, 2007.

[4] http://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html [retrieved: January, 2012]

[5] Michael Wooldridge, "Introduction to Multi Agent Systems", The MIT Press, 1999.

[6] Yariv Aridor and Danny B. Lange, "Agent Design Patterns: Elements of Agent Application Design", In proceeding of the 2th ACM international conference on Autonomous Agents (Agents '98), p. 108-115, 1998.

[7] Mohammed Eshtay, "Hierarchal Traveling Design Pattern for Mobile Agents in JADE Framework", 4th International Conference On Information Technology (ICIT), 2009.

[8] Ajay Kr. Singh, Ravi Sankar , and Vikram Jamwal, "Design Patterns for Mobile Agent Applications", Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices, Bologna 2002, http://autonomousagents.org/ubiagents/2002/papers/papers/22.pdf, [retrieved: January, 2012].

[9] Wenjuan Wang, Weihui Dai, Weidong Zhao, and Tong Li, "Research on Mobile Agent Systems for Agile Supply Management", Journal of Software, vol. 6, no. 8, Agust 2011, pp.1498-1505.

[10] Zakaria. Maamar, and Paul Labbé, "Moving vs. inviting software agents: what is the best strategy to adopt?" Communications of the ACM, Volume 46, Issue 7 (July 2003), pp. 143 – 144.

[11] Yasuyuki Tahara, Akihiko Ohsuga, and Shinichi. Honiden, "Agent system development method based on agent patterns", In Proceedings of the 21st international conference on Software engineering. IEEE Computer Society Press, 1999, pp. 356-367.

[12] Ananta Charan Ojha, Sateesh Kumar Pradhan, and Manas Ranjan Patra, "Pattern-Based Design for Intelligent Mobile Agents", 4th International Conference on Innovations in Information Technology (IIT '07), p. 501-505, 2007.

[13] Nicholas R. Jennings, "An Agent-Based Approach for Building Complex Software Systems", Communications of the ACM, 44(4), April 2001, pp.35-41.

[14] Danny B. Lange and Mitsuru. Oshima, "Seven Good Reasons for Mobile Agents", Communications of the ACM, 42(3), March 1999, pp. 88-89.

[15] Rahul Jha and Sridhar lyar, "Performance Evaluation of Mobile Agents for E-Commerce Applications", In Proc. Of the 8th Int. Conf: on High Performance Computing (HiPC 2001), LNCS, vol-2228, 2001, pp. 331-340.

[16] S.R. Mangalwade, K.K Tangod, U.P. Kulkarni and A.R. Yardi, "Effectiveness and Suitability of Mobile Agents for Distributed Computing Applications", In Proc. Of the 2nd Int. Conf: on Autonomous Robots and Agents, Dec 13-15, New Zealand, 2004.

[17] Elizabeth A. Kendall, P. V. Murali Krishna, Chirag V. Pathak, and C. B. Suresh, "Patterns of Intelligent and Mobile Agents", In Proc. of the 2nd Int. Conf: on Autonomous Agents, ACM Press USA, 1998, pp. 92 - 99.

[18] Nuno Meira, Ivo Conde e Silva, and Alberto Silva, "A Set of Agent Patterns for a More Expressive Approach", In Proc. of the 5th European Conf: on Pattern Languages of Programs (EuroPLoP2000), 2000, pp. 331-346.

[19] Sylvain Sauvage, "Design Patterns for Multiagent System Design", In Proc. of 3rd Mexican Int. Conf: on Artificial Intelligence (MICAI 2004), Mexico City, LNCS, Springer Berlin, 2004, pp.352-361.

[20] T. Tung Do, Manuel Kolp, and Alain Pirotte. "Social Patterns for Designing Multi-Agent Systems", In Proc. Of the 15th Int. Conf: on Software Eng. And Knowledge Eng. (SEKE 2003), USA, July 2003, pp.103-110.

[21] D. Deugo, M. Weiss, and E. Kendall, "Reusable Patterns for Agent Coordination", Coordination of Internet Agents: Models, Technologies, and Applications, Springer-Verlag, 2001, pp. 347-368.

[22] Valentas Daniunas, Vygintas Gontis, and Aleksejus Kononovicius, "Agent-based Versus Macroscopic Modeling of Competition and Business Processes in Economics", The Sixth International Multi-Conference on Computing in the Global Information Technology (ICCGI), Luxembourg, pp. 84-88, 2011.

[23] https://tao.uab.cat/ipmp/files/README [retrieved: January, 2012]

 ISBN: 978-1-61208-202-8