2011 Workshops of International Conference on Advanced Information Networking and Applications

A JADE Middleware for Grid inter-cooperated Infrastructures

Stelios Sotiriadis[1,2], Nik Bessis[1,2], Ye Huang[3], Pierre Kuonnen[4], and Nick Antonopoulos[1]
[1]School of Computing & Maths, University of Derby, United Kingdom
[2]Department of Computer Science and Technology, University of Bedfordshire, United Kingdom
[3]Department of Informatics, University of Fribourg, Switzerland
[4]Department of Information and Communication Technologies,
University of Applied Sciences Western Switzerland
[2](stelios.sotiriadis, nik.bessis)@beds.ac.uk, [1](n.bessis, n.antonopoulos)@derby.ac.uk
[3]ye.huang@unifr.ch, [4]pierre.kuonen@hefr.ch

Abstract— **During the past few years much effort has been put into developing interoperable grid models suitable of defining a decentralized control setting. Such environments may define new rules and actions to internal Virtual Organisation (VO) members and therefore posing new challenges towards to an extended cooperation model of grids. Particularly, VO members' knowledge may be expressed in the form of intelligent agents thus providing a more autonomous solution of communicating members. Herein we present a mobile agent middleware for Grid interoperable infrastructures. Facing the enlarging scale of Grid, the proposed middleware aims to extend the knowledge of a specific neighbouring of Grid members (VO) in relation to the addresses and the physical resources of known and unknown nodes which may join the Grid VO. The internal data are structured in a rational sequence and stored within a public profile of each member called metadata snapshot profile. The middleware is designed by employing the Java Agent Development (JADE) framework in which mobile agents are travelling throughout the domain and by collecting and updating internal data they extend the size of the VO. The interoperable standard is achieved by using the Critical Friends Community (CFC) model, as the mean to fulfil the inter-cooperation model.**

Keywords: Grid Computing, Critical Friends community, Intelligent Agents, Migration Mobile Agents, Java Agent Development Framework

I. INTRODUCTION

As Grid computing is defined a synchronized effort of multiple resources, which are typically inter-connected in loosely coupled random topologies, for solving a scientific or technical problem [1]. Those resources constitute a neighbouring of interacted members called a Virtual Organisation (VO). This form of distributed computing tends to be distinguished from the conventional systems mainly based on the heterogeneous attitude of their members. Parallel to Grid, Intelligent Agents offer an autonomous environment of problem solvers capable of self-directed actions in flexible environments. The Grid notion alongside with the Intelligent Agents is mutually following a relevant aim; to assist in the consolidation of an open distributed environment, even though a different perception. On the one hand, Grid has focused on the development of an interoperable infrastructure within dynamic VOs. On the other hand, Agents have focused on the realization of methods and techniques for autonomous acting members within uncertain and flexible domains. Motivated by the convergence of interests, we considering the need for dynamic ability of VO members for achieving self-sufficiency, and we suggest that an autonomous solution of individuals may be a step forward to an open grid infrastructure. The mean to achieve self-sufficiency is the mobile agents' paradigm; with the ability to act in response to a member's requirements whilst also learning from their operational environment [6].

Given this background, the work herein addresses a notable case, namely how Intelligent Agents may assist the resource discovery process within an uncertain Grid environment. Notable, the view here is to present the design and the functionality of a Grid middleware which is transparent to the VO members by providing them a proactive and reactive attitude. The model appreciates that the middleware design is based upon the Mobile Agents notion which allows software and data to be migrated from one member to another. Eventually, the mobility intelligence of the members will be supportive to the resource discovery phase. By performing migration of a members' internal knowledge contained within the metadata snapshot profile to any interacted VO node we aim to periodically update the awareness of members about their acting area.

In the following sections we present the motivation of the proposed study (Section II) and the related works (Section III). After, we discuss the methodology and the design strategy of the proposed middleware as well as the Foundation for Intelligent Physical Agents (FIPA) standard [9] (Section IV, V). The rest of the paper is organised as follows. First, we present the functionality of the migration mobile agents (Section VI). Then, we conclude our study by presenting an experimental study and results from the performance of the real time environment (Section VII). At last we conclude our work by discussing the future work part and the related challenges (Section VIII).

II. MOTIVATION

The study herein extends the work presented in [6], in which the resource discovery process is based on an ad hoc strategy. More particularly, each participant (node) acts as an

978-0-7695-4338-3/11 $26.00 © 2011 IEEE
DOI 10.1109/WAINA.2011.133

individual, detached from any centralized topologies articulated by the VOs. The method is extended to an inter-collaborative model wherein discovery is derived from data extracted from a member metadata snapshot profile. The primary challenging goal in building ad hoc Grid is supplying each Grid member with specific directions for continuously maintaining information related to each community participant [5, 6]. Such information is stored at each VO member public profile and is available for advertising on the resource discovery process. To define these models it is essential to share a common understanding of the structure among community members. Thus, we are focused and address the requirements that need to be announced and met from each VO participant in order to have a successful interaction. However, due to the huge number of different prerequisites which may be posed by the VOs members, we propose that it is essential to analyse and characterise the minimum requirements. That information can be constructed within the metadata snapshot profile in a generic and rational sequence. More specifically, we put forward a solution of structuring the profile as a four layer arrangement as described in [6] which contains policies and trust issues management (1st Layer), domain knowledge coupling (2nd Layer) and finally physical resources (3rd Layer) and times constraints (4th Layer) announcements.

In the same direction, internal knowledge and capabilities may be expressed in the form of intelligent agents thus providing a more autonomous solution of inter-communicating members [5, 6, 8]. Based on that fact we suggest an interoperable mobility agent model that performs migration to any interacting VO member and by travelling within each domain allows the discovery of resources dynamically [6, 7]. The originality of our approach is the mobility mechanism based on travelling and migration of software which utilizes the metadata snapshot profile and stores useful information during the route to each visited individual (node). Moreover, we suggest that resource discovery is a systematic and continually updating process that occurs directly within a VO. Finally, the planned solution includes an iterated route of travelling in which internal capabilities and knowledge are spread across the VO. However, due to the large number of inter-connected nodes resource discovery mechanism requires an effective and efficient amount of time that we call interval time. Consequently, we also consider this parameter in our middleware design.

III. Related Works

The dynamic ability of agents and the heterogeneous nature of grid make resource discovery a challenging issue. A number of authors describe the applicability of autonomic agents in grid environments [9,10] by suggesting that the grid needs agents as they form key components for a successful interoperable infrastructure. More specifically, authors in [11] suggest that software agents are programs that act on behalf of people and can accomplish a task by acting independently of the supervision of the user. By providing the ability to transport themselves between different systems, they can carry internal information which was obtained by each visited member. Finally their proactive and reactive nature makes them an ideal choice for a large distributed environment [12]. Within the scope of resource discovery, [13] proposes that various approaches exist for achieving discovery in grid environments such as the Query based and the Agent based approach. The Query based approach which is the most commonly used allows the resource information to be queried for availability. Conversely, in the Agent based approach agents can passively monitor and distribute information periodically or in response to another agent. The major difference between the aforementioned methods is that an agent is acting on its own decisions by using their internal logic in conjunction with the Query based approach which resides within a fixed query engine. Related works have utilized the agent framework to achieve the resource discovery of interoperable nodes. In [14] a web service agent has been proposed to simplify an interoperable model in which legacy web service components may have access to the agent system, thus encouraging interoperation. On the other hand [14] suggests that resource discovery is driven by autonomous multi-agents of the semantic grid and a standard interoperable web service has been addressed to some degree. Finally, the authors of [4] propose a grid service mobility integration of agents that enables the combination of the characteristics and the functionality provided by the agent paradigm, with the standardization provided for grid services. In the same direction the authors of [13] recommend that current agent based systems are immature and few truly agent-based systems have been developed, e.g. the Foundation for Intelligent Agents Framework (FIPA) [9].

IV. The Middleware Design

As discussed previously FIPA provides a standardization agent model for an interoperable environment. An integration mechanism of this consortium is the Java Agent Development Framework (JADE). Originally, JADE aimed at developing multi-agents systems and applications conforming to FIPA standards for intelligent agents [2]. JADE has been fully coded in Java and was the programming language of choice because of its many attractive features, mainly geared towards object-oriented programming in distributed heterogeneous environments. Thus, in this chapter we describe the design of the proposed middleware which creates mobile agents for achieving resource discovery in distributed environments.

More specifically, each node contains a JADE middleware which creates two agent platforms, the server platform service, and the client platform service. On the one hand, the server which is selected randomly is responsible for creating an agent for performing migration to interacted members by utilizing the metadata snapshot profile information and specifically the IP addresses of the known members. That information (IP addresses) is stored within the snapshot profile and is subject to the agent intelligence for collecting and updating after a successful route. The agent server creates a service container which called main container and it is the one that implements the mobile agent. The main container functionality is to first access the

snapshot profile with the IP addresses of the well known members and starts the trip by moving the entire code to the first member of the list. Also, the main-container collects the physical resources including operating system architecture, total memory, free memory, free physical resource memory, total physical resource memory, amount of free swap space and total amount of free swap space of the server node. Finally the migration agent starts the trip and visits the first member in the list for requesting information about IP addresses and physical resource information.

On the other hand, the client platform creates an agent container service capable of generating an agent for receiving and sending internal knowledge to the migration agent coming from the server. More specifically, the client container creates an agent waiting for a stimulus from the server agent. Its functionality is to itinerary waiting for a Server message aiming to collect the IP addresses from the snapshot file as well as the physical resources of the client which are structured dynamically. As a result, when the migration agent visits the client, both exchange information on the subject of the IP addresses as also the physical resources. After a successful interaction the migration agent leaves the first client node and migrates directly to the second node without returning back to the starting point. The same sequence of actions happens in the next nodes and the procedure continues. Finally, when the migration agent visits the last node of the list it will return back to the starting point which is actually the creator of the migration agent service.

V. THE AGENT SERVICE SPECIFICATION

In most today's agent systems, migration of agents requires flexibility and heterogeneity in the agent platform that creates the agent. In the first case of the interacting agents the intra-platform functionality offers a more centralized topology of agents. More specifically, a member is selected as the host and creates a platform referred to specific agent service. Members belonging to the same domain are aware of that platform and they generate sub-platforms which refer to the previously created host platform. On the other hand, inter-platform agents offer a decentralized model which creates platforms dynamically for each member without having knowledge about the other members' platform settings. In such environments each domain participant contains a different platform which is created locally and generates an agent. Agent functionality involves waiting for requests from other agent platforms in order to exchange internal knowledge. In other words, any of the agents are capable of performing communication directly so a new service can be created dynamically.

In this study the mobile agent strategy is based upon the FIPA specification. The middleware is based on the Java Agent Development Framework (JADE) [2], which is software implemented in Java and simplifies the implementation of multi-agents systems. More specifically, each VO member contains the JADE middleware which is capable of creating mobile agents with the ability to behave as migration agents to any interacting members within the same or different VO. Furthermore, we assume that members of VOs contain the middleware for creating mobile agents. The members' platform acts as the middleware in which agent containers can be created, so that inter-platform behaviour offers a decentralized model of interacting members. It is fundamental that each mobile agent consists of three parts, the *code*, *state* and *data*. The *code* is the part of the agent that migrates to a different platform, the *state* is the execution environment and the *data* is that aspect that consists of members' variables such as the internal knowledge.

Our sample consists of three members that are able to communicate with each other. A node is selected to be the host, which in our case will be the creator of the agent service. The remaining nodes are capable of creating a sub-platform specification which refers to the Host member platform. In other words, sub-platforms accept communication from an agent "*Agent A"*, while an internal agent waits for the connection. The agent starting from the host platform traverses a route to each node to collect and update the internal information of visited members, and then returns back to the host. The service can be repeated by any of other members, however each time other members should be alerted of the service creator address. The aforementioned scheme illustrates the inter-platform communication model; in which agents are created dynamically by the inter-platform utility service. The mobile agent migrates to different platforms and exchanges information with local agents. This solution predisposes the need for compatibility between different platforms and security issues needs to be resolved by the agents. Considering inter-platform necessarily, the implementation of the middleware is achieved by using a specific add-on feature of JADE for developing inter-platform services [2].

VI. THE MIDDLEWARE FUNCTIONALITY

In this section we present the functionality of the middleware service by discussing the server and client features.

A. The server

As mentioned in before the middleware is designed to behave either as a server or as a client. The server part implements the mobile agent class which creates the migration behaviours of the agent. More specifically the middleware implements five cases as follows:

Case 1: The class collects the addresses of the trusted members by utilizing the metadata snapshot file which contains the IP addresses.

Case 2: A mobile agent instance sends an ACL message to the client agent.

Case 3: The server waits a response from the client agent. In any case the migration agent moves to the next node by creating an itinerary of addresses based on the metadata snapshot profile data.

Case 4: The migration agent assigns an id to each newly discovered host which it's actually the IP addresses of the members.

Case 5: The migration agent itinerary is finished and the new addresses are stored within the metadata snapshot profile.

Hence, during the route the migration agent moves from member to member and continues its execution from the current case point. Notable, the middleware implements a strong migration mechanism which means that agent continues execution at the new resource, in contrast with the weak migration in which execution starts from the beginning.

B. The client

The agent client creates a platform for initializing the agent cyclic behaviour. More specifically, the internal agent of the node initializes the behaviour for receiving and sending information from the travelling agent. Furthermore the code implements three cases as follows:

Case 1: The agent gets the IP address and the physical resources of the node.

Case 2: The agent is ready for receiving a stimulus connection from the travelling agent. In this case the client goes to a sleep mode until without overloading the client physical resources.

Case 3: In the case of a forthcoming ACL message from the server the client collects the metadata snapshot profile data and interacts with the migration agent. The addresses and the physical resources are stored within a vector of the forthcoming mobile agent.

So, after a successful interaction the agent status is idle (sleep) and it is waiting for connection from any mobile agent. It should be mentioned that the mobility service supports the inter-platform configuration which allows information exchanging from different migration agents.

VII. THE JADE MIDDLEWARE ARCHITECTURE

As presented in aforementioned sections the JADE middleware creates a server and a client component respectively. Both functionalities offer novel intelligent features, however the server part is principally the concept behind the mobility of the middleware. So, here we discuss the architecture of the server part which is mainly based upon four different components namely as follows:

1. The Main Controller Agent

The *main controller* it is actually a Java based application which:

a) Initializes the JADE environment

b) Initializes the Controller Agent Component

2. The Controller Agent

The *controller agent* is responsible for collecting the IP addresses from the metadata snapshot profile of the Server. Consequently, its functionality contains the following procedures:

a) Initializes and sends a message to the *receiver message agent* component which contains the number of the agents that are about to be created from the *controller agent*. It is noteworthe to mentioned that the server part could be able to generate several mobile agents at the same time.

b) Through an iterated loop creates the agents with respect to a simple function. More specifically, the function is defined as: Agent number = Agent number * 0,1. So, for a sample of 100 nodes the controller agent will be able to create 10 agents which are about to be migrated as discussed in the next step.

3. The Mobile Agent

The *mobile agent* component is initialized from the agent controller and accepts as the main parameter a vector with the hosts which is about to visit. The functionality contains the *setup()* and *action ()* methods as follows:

a) The *setup()* method

Initializes variables and creates a new behaviour which will be iterated in a cyclic mode within the method *action()*

b) The action() method

The *action()* method contains a state variable that accepts the values 0 and 1. While the state is 0 moves it self to the next IP address of the vector. In the case that state is 1 moves it self to the first IP which is the starting point, which means that the visiting route is completed. The important point of the agent intelligence for performing several behaviours is controlled within the *action()* method. So, in this part we initialize the before and after behaviour of an agent visit. In both situations the JADE middleware creates two behaviours that are about to be executed at a specific time. More specifically the behaviours are:

- The *BeforeMove()* behaviour

On the one hand, the BeforeMove() is responsible for performing actions prior to an agent starting move. In our middleware the BeforeMove() displays information messages.

- The *AfterMove()* behaviour

On the other hand, the AfterMove() behaviour is executed when the agent leave the first node. Its functionality includes the following actions:

a) Sends a message to the client agent and requests for new IPs and new physical resource data.

b) Waits within an iterated loop for a response from the client agent for a specific time-out value.

c) Sends the information which remotely collected from the client agent to the Receiver Message Agent. In the case of a time-out, sends a failure connection message.

d) Finally, when the entire procedure is finished the agent delete it self and release the client. Then the client goes to a non overloaded sleepy mode.

4. The Receiver Message Agent

The *receiver message agent* functionality is to act as the message bus of the middleware. More specifically the component accepts a message from the *controller agent* only one time per route. The message contains the names of the newly created client agents from the *controller agent*. Following to that, accepts several messages from the mobile

agents which contain the old and newly found IPs as well as physical resource information. Figure 1 illustrates the JADE server middleware architecture.

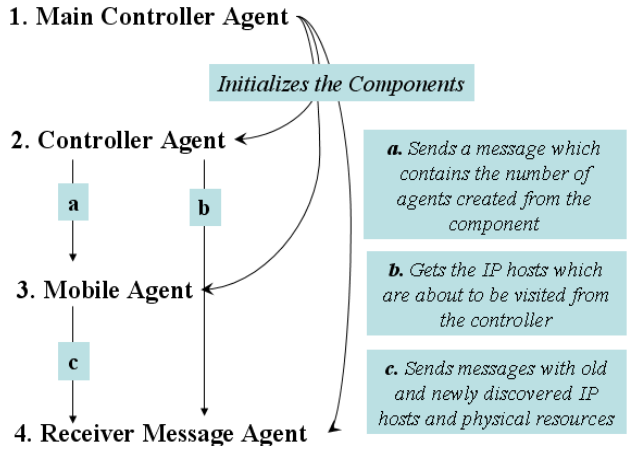


Figure 1. The Server Middleware Architecture

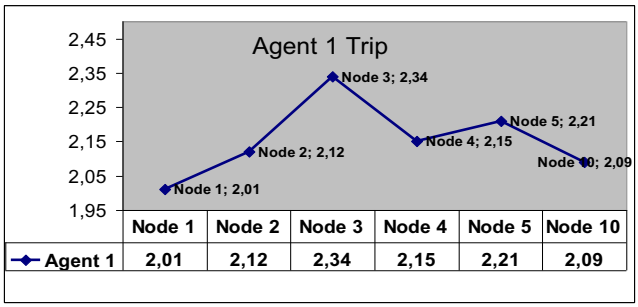
VIII. Experiments and Results

In this section we present a real-time environment experiment of the JADE middleware functionality. More specifically, we present two experiments and evaluation of results as follows:

1. In the first evaluation test we describe the resource discovery between two small scale VOS with total size of ten nodes. It should be mentioned that VO_1 IP addresses are those starting with 192.168 and VO_2 addresses starting with 195.168. We have installed the JADE middleware to each of the ten nodes and the agent metadata snapshot profile. So, in this experiment, we have selected randomly that $Node_1$ will be the server machine which starts the resource discovery process. The aim here is to achieve resource discovery of the VO_2 nodes. For that reason we assume that $node_1$ and $node_5$ are Critical Friends Community (CFC) members as discussed in [3]. More specifically CFC promotes an inter-collaborated threshold delivery service among Virtual Organisation (VO) members (nodes). Thus $node_5$ will be able to redirect the process to VO_2 nodes which are currently unknown to $node_1$.

Starting from the $Node_1$ (the server) a travelling agent is initiated which access the snapshot profile and collects the addresses stored within the profile. In out sample the travelling agent first communicates with $node_2$ (the first client in the list). More specifically, the mobile agent stimulates a connection with the client agent which offers an inter-platform utility service. After that, we assume that a security interaction occurs in which both members exchange security certificates. It should be mentioned that the JADE runtime environment contains security add-on validation features, however we currently assume that extra VO authorization measures may occur. Then, the travelling agent of $Node_1$ requests the metadata snapshot addresses of the $Node_2$. The last one sends the addresses to the $Node_1$ agent and both members update their agent property files. Then the mobile agent moves to the next $node_3$ and so on. Lastly, agent visits the $node_5$ and a new address is discovered. The new data are collected and the agent continues the trip. As $node_5$ is the last node in the list it returns back to the starting point and updates the data of $node_1$. In the next trip $Node_1$ mobile agent will visit $node_2$, $node_3$, $node_4$, $node_5$ and $node_{10}$ respectively. So starting from $node_2$, the travelling agent updates the profile and stores the new address of $node_{10}$. The procedure continues in the same order for the next $node_3$. Finally the agent will migrate to the interconnected member $node_{10}$ and after a successful interaction the new addresses of ($node_2$, $node_3$, $node_4$ and $node_5$) are stored in the profile. Also the addresses of $node_6$, $node_7$, $node_8$ and $node_9$ are collected and returned back to the starting point. So, after a number of iterations of the travelling agent both domains VO_1 and VO_2 will be able to extend their knowledge about the addresses of interconnected members. It should be mentioned that after a successful interaction mobile and client agents exchange physical resources. We have run some experiments in our testbed, and we have discovered that a mobile agent needs approximately *2,25* seconds to move from machine to machine. So the diagram 1 presents the interval times from the agent trip. As we can see the highest interval time is *2,34* seconds and the total interval time of a complete route is *13,14* seconds.

DIAGRAM I. VO1 Agents Performance



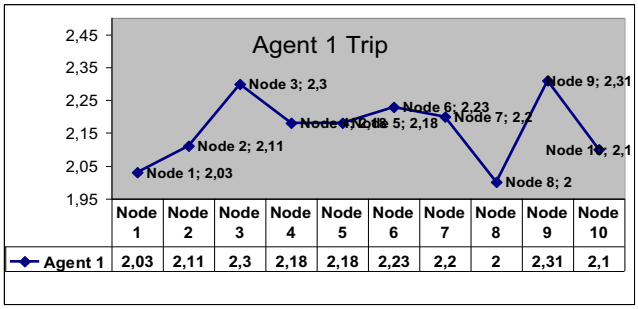
More specifically, as we can see from the diagram the highest interval time is *2,34* seconds. In this case the total interval time is considered as follows:

Trip 1: The first route as described in diagram 1 in which the agent visits $node_1$ to $node_5$ and $node_{10}$.

Trip 2: During the second route the agent visits $node_1$ to $node_{10}$.

After the discovery of the new members based upon the CFC behavior of node5 and node8 the new times of the agents are presented in diagram 2.

DIAGRAM II. VO1 and VO2 Agents Performance



Therefore, after the evaluation of the results we conclude that the total interval time of the route is *34, 56* second. It is obviously that if we consider a huge number of nodes the total interval time will be massive. For that reason we have employed a decomposition model of agents as described in [6]. In this model we avoid to enlarge the nodes metadata snapshot profile by clustering the nodes and assigning agents to specific groups of nodes.

2. In this section we have run a test within an environment of 200 nodes. For that reason we have decided to generate 20 agents according to the function described in Section VII. Each one of those agents will be capable of performing migration to ten nodes. More specifically agent1 will be moved to node1 to node10, agent 2 will be moved to node11 to node20 etc. Thus, diagram 3 illustrates the times of the agent trip. As we can see from the diagram the highest visit time is for agent 4, which requires *2,*4 seconds to visit node31. So, in experiment 1 the highest agent time for a complete trip is *34,56,* which is identical to experiment 2 agent 4 trip of *34,62* seconds. It is obvious that the average interval time for discovery among 200 nodes it is quite similar, in other words by decomposing VOs to smaller parts we can achieve a quality threshold optimization of the interval time. However, a delay time of 4 seconds occurs when the agents are arriving back to the server. It also should be mentioned that in the previous sample of 200 nodes we have collected 50 new addresses (extracted from the snapshot profile of each member), so in the next route we will generate 25 agents and the total interval will be identical to the first example of 10 nodes discovery. So, the main contribution of our work is undoubtedly the fact that the discovery time of 1 agent for 10 nodes is identical to the discovery of 20 agents for 200 nodes. Subsequently, the main endeavour is to empower the resource discovery by decomposing VOs to smaller parts.

IX. Conclusion And Further Work

In the work herein we consider the need for dynamic ability of VO members, so we suggest that an autonomous solution of individuals may be a step forward to an open grid infrastructure. The above sections have shown that a convenient way to achieve self-sufficiency of Grid nodes is the mobile agents' paradigm; with the ability to act in response to a member's requirements whilst also learning from their operational environment. The model provides an interoperable resource discovery concept by introducing a novel functionality; in which several members of different VOs are capable of communicating with each other in order to extend VO boundaries. Nevertheless, by decomposing the VO into smaller parts we can achieve a significant low interval time.

X. References

[1] I. Foster, C, Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of High Performance Computing Applications 15(3), 200, 2001.

[2] F., Bellifemine, G., Caire, A., Poggi, G., Rimassa, "JADE: A software framework for developing multi-agent applications", Lecture Notes in Computer Science, Intelligent Agents VII Agent Theories Architectures and Languages, pp. 42-47, 2001, Springer Berlin / Heidelberg.

[3] Y., Huang, N., Bessis, A., Brocco, S., Sotiriadis, M.,Courant, P., Kuonen, B. Hisbrunner, "Towards an integrated vision across inter-cooperative grid virtual organizations". Future Generation Information Technology (FGIT 2009), pp.120-128, Springer LNCS, Jeju island, Korea.

[4] L., Moreau, A., Avila-Rosas, V., Dialani, S., Miles, X., Liu," Agents for the Grid: A Comparison with Web Services (part II: Service Discovery)", Proceedings of Workshop on Challenges in Open Agent Systems, Italy, pp. 52-56, 2002.

[5] S., Sotiriadis, N., Bessis, P., Sant, C., Maple. (2010). "A resource discovery architecture of loosely coupled grid inter-cooperated Virtual Organisations using mobile agents and neural networks", Fifth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC 2010), Fukuoka Japan, 4th-6th November 2010 (to appear).

[6] S., Sotiriadis N., Bessis Y., Huang P., Sant C., Maple, "Defining minimum requirements of inter-collaborated nodes by measuring the heaviness of node interactions". International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010), IEEE, Krakow, Poland, February 2010.

[7] S., Sotiriadis N., Bessis Y., Huang P., Sant C., Maple, "Towards to decentralized grid agent models for continuous resource discovery of interoperable grid Virtual Organizations", The third international conference of Applications and Digital information and Web technologies (ICADIWT), Instabul, Turkey, July 2010.

[8] S., Sotiriadis N., Bessis P., Sant C., Maple, "Encoding minimum requirements of ad hoc inter-connected grid virtual organisations using a genetic algorithm infrastructure", IADIS multi conference on computer science and information Systems (MCCSIS 2010), Freiburg, Germany, July 2010.

[9] The Foundation for Intelligent Physical Agents (FIPA), "http://www.fipa.org".

[10] I., Foster, N.R., Jennings, C. and Kesselman, "Brain meets brawn: why Grid and agents need each other", In: 3rd International Conference on Autonomous Agents and Multi-Agent Systems, 2004, New York, USA. pp. 8-15, 2004.

[11] P. D., Cox, Y., Al-Nashif, S., Hariri, "Application of Autonomic Agents for Global Information Grid Management and Security", in: Summer Computer Simulation Conference 2007 (SCSC 2007), San Diego, USA, 2007.

[12] K.G., Zerfiridis, and H.D., Karatza, "Mobile Agents as a Middleware for Data Dissemination". Neural, Parallel & Scientific Computations. Dynamic Publishers, Vol. 10, No. 3, pp. 313 323, Atlanta, September 2002.

[13] K.G., Zerfiridis, and H.D., Karatza, "File Distribution Using a Peer-to-Peer Network - A Simulation Study". Journal of Systems and Software, Elsevier, Vol 73/1 pp. 31-44, 2004.

[14] T.E., Athanaileas, N.D., Tselikas, G. V., Tsoulos, D. I., Kaklamani, "An agent-based framework for integrating mobility into grid services", Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications, Innsburck, Austria, 2008.