Implementation of Agent based Dynamic Distributed Service

Prof. I.Ravi Prakash Reddy irpreddy@gnits.ac.in
IT Dept.
G.Narayanamma Institute of Tech & Science
Hyderabad-500008

Dr. A.Damodaram adamodaram@jntu.ac.in
Prof., Dept. of CSE
JNTU College of Engg.Hyderabad

Abstract

The concept of distributed computing implies a network / internet-work of independent nodes which are logically configured in such a manner as to be seen as one machine by an application. They have been implemented in many varying forms and configurations, for the optimal processing of data.

Agents and multi-agent systems are useful in modeling complex distributed processes. They focus on support for (the development of) large-scale, secure, and heterogeneous distributed systems. They are expected to abstract both hardware and software vis-à-vis distributed systems.

For optimizing the use of the tremendous increase in processing power, bandwidth, and memory that technology is placing in the hands of the designer, an agent based Dynamically Distributed Service (DDS, to be positioned as a service to a network / internet-work) is proposed. The service will conceptually migrate an application on to different nodes**.** In this paper, we present the design and implementation of an inter-mobility (migration) mechanism for agents. This migration is based on FIPA[1] ACL messages. We also evaluate the performance of this implementation by using a Distributed framework.

Keywords: Distributed Systems, Agents, Agent Migration

1. INTRODUCTION

Over the last two decades, the concept of distributed computing has been implemented in varying configurations and on diverse platforms. In current context, a distributed system implies a networked system of nodes in which a single application is able to execute transparently (and concurrently) on data that is, (or may be) spread across heterogeneous (hardware & operating system) platforms. The salient payoffs of distributed computing may be listed as:

- Enhanced performance (in respect of both speed up and scale up).
- Resource sharing (in respect of data as well hardware and software processing elements).
- Fault tolerance and enhanced reliability.
- Serve as the basis for grid computing.

Several other relevant issues while assessing the relevance of distributed computing vis-à-vis the current computing environment and this paper are:

- Interoperability in a heterogeneous environment will continue to be the dominating theme in future applications.
- Communication technology advances will continue to fuel the need for more bandwidth and enhanced Quality of Service specifications.
- The rate of increase in data processing and storage is greater than that of data transfer.
- Most users are reluctant to switch platforms, as are developers to switch technology paradigms.
- The individual behavior of the vast number of interconnected nodes based on individual workstations mandate that any service acting upon them universally must be dynamic in nature.
- In many computer installations /complexes, a lot of state of the art hardware is not used round the clock. There are times when it is idle, or is under-utilized. When this happens, it may be used by other applications for processing purposes remotely. Networking enables this. Inter-networking further emphasizes the same.

In view of the above, it is forecast that distributed processing of applications and data will no longer be restricted to high end research and scientific applications, but will become as normal as any other service provided over an inter-network. The Internet and the Web themselves are a viable distributed computing domains. Distributed computing however, has yet to gain the type of proliferation mandated by enhanced rates of data processing as well as transfer.

To further the optimization of internet-works (including networks), by the use of distributed computing concepts, a Dynamically Distributed Service, analogous to e-mail, FTP, Voice over IP, etc, is proposed, which can be made available on demand, in an intranet/inter-network. The service will conceptually migrate an application on to different nodes. In this paper, we have presented a proposal for the mobility of agents between various agencies, based on the agent communication language (ACL) proposed by FIPA. This Dynamically Distributed Service (DDS) is at variance with distributed paradigms used till date, though *no changes to the underlying hardware or OS are proposed* in its implementation.

The efficient utilization of network resources is an important issue. The problem is hard due to the distributed nature of computer networks, high communication demands and the desire for limited communication overheads. One solution to such challenges is to design efficient, decentralized and fault-tolerant data propagation model which accomplishes tasks with no access to global network information. Mechanisms of agent propagation are useful because agents can be organized into efficient configurations without imposing external centralized controls. Operation without a central coordinator eliminates possible bottlenecks in terms of scalability and reliability.
In the section 5,we evaluate the performance of DDS by applying it to distributed calculation of Prime numbers.

2. DISTRIBUTED AGENTS

The research areas of multi-agent systems and distributed systems coincide, and form the research area of *distributed agent computing*. Multi-agent systems are often distributed systems, and distributed systems are platforms to support multi-agent systems[2].

Agents are considered to be autonomous (i.e., independent, not-controllable), reactive (i.e., responding to events), pro-active (i.e., initiating actions of their own volition), and social (i.e., communicative). Sometimes a stronger notion is added (beliefs, desired, intentions) realizing intention notions for agents. Agents vary in their abilities; e.g. they can be static or mobile, or may or may not be intelligent. Each agent may have its own task and/or role. Agents, and multi-agent systems are used as a metaphor to model complex distributed processes.

Both distributed systems and agents share the notion of 'distributedness'. The area of multi-agent systems addresses distributed tasks; distributed systems addresses supporting distributed information and processes.

The area of *distributed agent computing* is the area in which both approaches intersect. Both can be synergized to further optimality. Mobile agents transfer code, data, and especially authority to act on the owner's behalf on a wide scale, such as within the entire Internet. Because of this advantage, we have decided to use mobile agents for migration.

3. PROPOSED MODEL

The platform chosen for implementing migration was JADE[3], because it is a widely adopted platform within the software agent development and research communities. It is open source and complies with FIPA specifications.

3.1 The JADE platform

The JADE platform is divided into a large number of functional modules, which can be placed into three categories in general terms:

Core. The core of the platform is formed by all components providing the necessary execution environment for agents' functioning.

Ontologies and Content Languages administration. This consists of the agency's mechanisms for carrying out information processing in ACL messages, and the internal structures that the agency and agents will use to represent this content.

Message transport mechanisms. Mechanisms and protocols used to send and receive messages at both intra-agency and inter-agency level.

At the core of the JADE platform is the concept of the container, which is the minimum execution environment necessary for an agent to operate. Each container in JADE is executed in a different Java virtual machine, but they are all interconnected by RMI (Remote Method Invocation).

Containers do not only enable groups of agents to be separated into different execution groups, but agencies may also be distributed in various machines so that each has one or several of them. One of the different existing containers is the principal, which represents the agent itself and which gives orders to all the others. JADE also provides mobility between containers. For this reason, if the agency is distributed in various machines, agents can move between them. However, accepting this type of mobility as migration could be considered a mistake." Satellite" containers are highly dependent on the principal and many operations carried out by the agents within them end up passing through the central node. Furthermore, the connections between them (carried out by RMI) must be permanent, as if not, many errors due to the loss of link may be generated. As we can see, using this type of mobility as a typical migration ends up making mobile agent systems' scalability disappear because a certain type of operations is centralized in a single node. However, it may be very useful to use the diagram of containers to distribute the processing of agencies that have to bear a heavy load or to isolate some agency types within a single agency for security reasons.

These details lead to the necessity for inter-agency migration, which is carried out through a non-permanent channel and makes a system of mobile agents available that is much more scalable, and in which agencies are totally independent units. This independence is not only desirable from the point of view of fault tolerance, but also because of privacy.

3.2 Our proposal for migration using ACL

The idea of creating a migration using ACL messages came from FIPA's specification regarding mobility, where this type of migration is proposed. However, as mentioned above, this specification only gives a general outline of the ontologies, the protocol, and the life cycle of a mobile agent. It has not been updated due to the lack of implementations and has become obsolete within the FIPA specifications. For these reasons, we have found that there is a need to propose extensions to the specification to cover situations that it does not deal with.

The design for a migration using ACL means that transmission of mobile agents between two agencies will be carried out using the message system between agents. In other words, the agent (both the code forming it as well as the state that it is in) will travel as the content of a message between two agents. Specifically, the agent will travel in the message between the AMS agents of each of the agencies involved in a migration.

Because the agencies have mechanisms for sending and receiving messages, using a parallel transmission protocol is not necessary. This is an advantage in interoperability terms and enables agents to be transmitted using the various message transport protocols (HTTP, HOP, SMTP, etc). Furthermore, this is achieved in a totally transparent way.

The first logical step in this process is to design the ontology and the protocol that will be used in the exchange of messages between the two agencies. This protocol has the movement of the agent as its final purpose. Defining an ontology basically consists of defining the elements that will form the content of an ACL message to give a common interface between the two parties when extracting the information of the message.

The two possible migration models that are proposed in the initial FIPA specification deal with one migration directed by the agent and another directed by the agency. In our implementation, we have decided to adopt the migration directed by the agency, which is more robust, as it enables the agency to decide which migration requests are accepted and which are not.

The ontology initially specified by FIPA is made up of seven elements: five concepts ("mobile-agent-description", "mobile-agent-profile", "mobile-agent-system", "mobile-agent-language", and "mobile-agent-os") and two actions ("Move" and "Transfer").

Of these concepts, we only use the first one, "mobile-agent-description". This is because it is very difficult to develop systems that enable agents with different architectures to migrate with total interoperability, at the current level of agent technological maturity. These agents could have been written in different languages or executed in different operating systems. For this reason, these concepts are never used, assuming that mobile agents which migrate move between the same agencies. Obviously, if the agency has been developed in a language like Java, a migration between agencies lodged in different operating systems is possible. However, this is a characteristic of this language which is transparent to the agency, and therefore does not involve the need to use the concept "mobile-agent-os", for example. In any case, although we do not use it, we maintain these concepts to ensure compatibility in case it is possible to make agents migrate between agencies implemented in a different way or with different languages.

The concept "mobile-agent-description", on the other hand, is highly useful to us. Within it are several fields that define the characteristics of the mobile agent in question. Among others, these include the characterization of the code and its data.

Of the two actions specified, we have only implemented "Move", for migrations directed by the agency. We do not take the "Transfer" action into consideration, which can be used for migrations directed by the agent, although it is supervised by the agency.

Once the content of the ACL messages was described, we moved on to enumerating the protocol by which the migration process is carried out. A diagram of the messages exchanged during the migration process can be seen in Figure1.

- Firstly, the agent wishing to migrate starts a conversation with the AMS agent of the local platform to make a request for migration. This request is the first step in the standard FIPA-Request protocol and consists of a Request-type message with a Move action and a *MobileAgentDescription (MobileAgentDescription* is the name of the class that implements the concept of "mobile-agent-description"), in which the code and data fields are empty.
- When the AMS agent receives the request for migration from a mobile agent, the first thing it does is to decide whether to accept it or not according to a given criterion. If the migration is accepted, the AMS agent sends *din Agree* message to the agent, or if not, a *Refuse* message.
- If the migration is accepted, the first thing that the AMS agent does is to obtain the code and data (serialized instance) of the agent that made the request and fills in the code and data fields of'the*MobileAgentDescription.*
- The next step is to make contact with the AMS belonging to the agency to which the mobile agent wishes to migrate. To this end, the local AMS must start a parallel conversation to that between the agent and the remote AMS.
- When the remote AMS receives the request, the agent's code and data travel within the *MobileAgentDescription* that the local AMS has prepared.
- Following its own criteria, the remote platform decides whether to accept the in coming agent. If so, it responds with a *Agree* message, and if the agent does not meet the requirements specified by the agency to execute it, it responds with a *Refuse* message.
- The remote AMS loads the agent's class, deserializes its instance and restores its execution. Once this entire process has been successfully completed, the standard FIPA-Request protocol is completed by sending an *Inform* message to the local AMS.
- The final step in the protocol consists of informing the agent that started the process. If the process has been successfully completed, the original agent is destroyed.

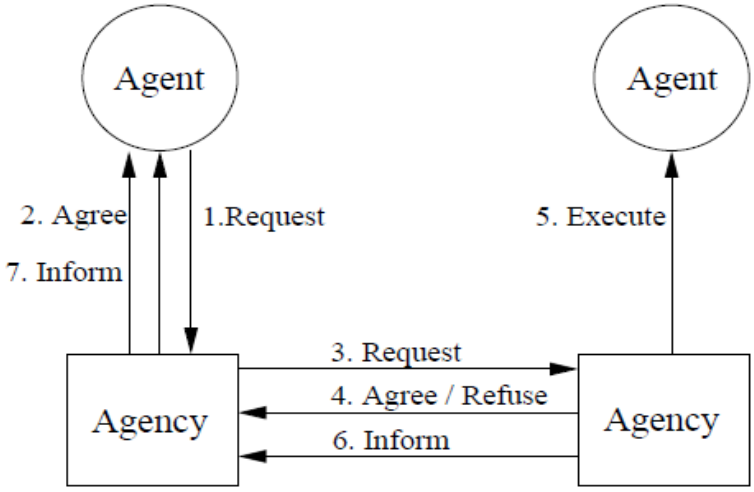

FIGURE 1: Exchange of ACL messages in the migration protocol

4. DESIGN COMPONENTS

After describing the ontology and the protocol used, the components necessary to carry out the entire process are listed below.

Migration Manager: The Migration Manager component is responsible for carrying out all operations enabling the execution of an agent to be suspended in the local agency. Also, it enables its state and code to be processed. These operation allow to insert the agent in an ACL message. In the remote agency, it is responsible for undoing the messages that contain agents, decoding them and acting together with the agency to restore the agent's execution.

Class Loader: The class loader is the component that constructs the representation of the class in the memory so that it can be used in the code. The bytecode of the class is extracted from the ACL message and loaded during the deserialization time of an incoming agent. At the same time, each classloader provides a space of different names from the others, so that two agents created with one class with the same name do not conflict within a single platform if they are loaded by different classloaders.

Mobile Agent: We have created the *MobileAgent* class, inheriting the properties of a basic agent and adding the functionality of being able to migrate to it. This functionality provides it with the *doMigrate(dest)* method, which starts the migration protocol when invoked.

Conversation Modules: These modules were implemented using the behaviours of the JADE model [4]. Behaviors represent agent tasks and are also a useful tool for designing the protocols that govern conversations using ACL. There are basically two components of this type developed to provide mobility. The first is the agent's behavior. This component is launched when the function *doMigrate(dest)* is invoked and is responsible for supervising the migration from the agent's point of view. The second was implemented within the AMS agent to help it administer its part of the migration protocol. This behavior has a double functionality as it was designed for playing the roles of the local AMS and the remote AMS. The most complex part of the implementation of this component is its functioning as a local AMS: parallel conversations (AMS-Agent and AMS-AMS) which depend on each other must be taken into consideration.

5. PERFORMANCE EVALUATION

Providing agents with abilities to migrate continues to be a highly challenging problem[5]. We conduct a set of experiments in two environments: 4 heterogeneous computers and 45 (almost) homogeneous computers. Specifically, we are looking for a way to find optimal configurations for migration in both environments. The conclusion from this work is that introducing propagation to a system in a form of agent migration in both networks could considerably decrease the execution time according to used algorithms and established assumptions.

5.1 The Problem

The prime number generation problem was selected for testing the DDS. We assume that we generate a given number of primes above a given start number and write the results into the file. All computers search prime numbers and as soon as possible they send the solution to the one, which is responsible for saving the results in a file. Prime number generator was selected to be solved in a distributed way because of its computation-focused nature. A range of numbers to examine is given and from the result point of view, it does not matter, on which machine the computation is running. The most important aspect is that there is a possibility to move agents from the slow computers to the faster ones.

The algorithm for prime number is simple. For each x from given range [a,b] calculate square root c = sqrt(x) and check if any number from range [2,c] divides x without reminder; if no then add x to the list of prime numbers. Of course, there are more efficient ways to calculate prime numbers. The goal is not to calculate them as fast as possible, but to show how distribution could be managed using agent migration.

This approach has several advantages. One of them is that the range to search the primes can be divided, and the results can be merged at the end giving the complete solution to the problem.

In order not to create a bottle-neck, partial results are being sent continuously to be saved into the file.

5.2 DDS Framework Architecture

In this architecture, in general the DDS framework consists of two kinds of units: nodes and a broker. A node is a component, on which agents reside, which is supposed to search prime numbers. A broker is a component, which distributes the task into the framework and saves the results into the file. It is not destined to calculate primes.

We assume the following algorithm for distributing the task over the nodes. When the broker gets the task order it knows how many nodes are available, because when a node enters the network, it sends a ready message to the broker. We do not consider any changes after the broker has received the task order. At the beginning there is no information about the available resources, so primes search range is divided equally by the number of nodes and then sent as a task request. While processing, nodes also send found primes to the broker by portions.

5.3 Main Process

The main process is the core of the framework[6]. Its main goal is to distribute task and afterwards collect all messages concerning reporting in order to display the final results. All activities used in the diagram are agent services.

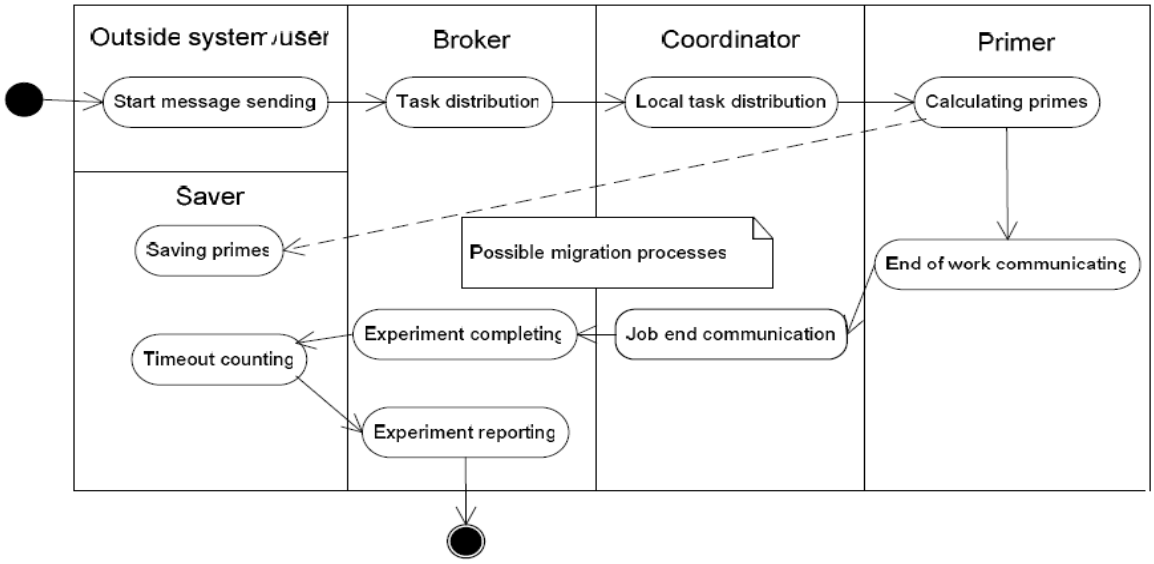

FIGURE 2: Main process diagram

There are 4 types of agents participating in the main process: Broker, Saver, Coordinator and Primer. This process, depicted on Figure 2, starts outside of the system, when a user or agent from another system sends a message to the Broker that includes the range of numbers to search primes from. The next step is done by the Broker, which distributes task to Coordinators. The diagram shows the control flow only for one Coordinator and one Primer, with actually many Coordinators and many Primers.

Task distribution from the Broker goes to the local level and finally each Primer gets its task. Searching primes consists of many single search processes after which results are sent to the Saver agent. This is shown in figure by a dashed arrow. The box "Possible migration processes" denotes the place in the main process flow when migration process is possible. They take place after the initial task distribution and before end of work.

The last phase of the main process starts when Primers finish searching the whole range of numbers they got. Each of them sends a message to the Coordinator (a Coordinator "commanding" the location where Primer resides) and when all Primers report back to Coordinators, agents send job end message to the Broker. When Broker gets all job end messages, it expects the main process to be nearing completion. Because there are possible message asynchronisms, Saver agent starts a timeout counting in order to receive all messages from Primers, whose results have not been written into the file so far. This possibility exists mainly because each result message includes large data to write. When timeout is over, Saver agent confirms that there are most probably no other result messages (also called save messages). When Broker gets this message, it displays information about the experiment. This ends the main process.

5.4 Migration Calculation

This process assesses the ability of a node to perform a task. The key point here is that the estimation is based on the previous efficiency of a node when performing a task. It is compliant to the assumption that at the beginning of an experiment the ability of nodes to perform the task is unknown.

When a Coordinator gets all progress reports it estimates remaining time till the end of experiment based on the data it has. This is an important moment in the whole algorithm. Time till the end is estimated based on a sample from the previous change in the environment for the node - from the last migration or the beginning of the experiment. Basically, when Primers are reporting, they deliver two parameters - what is the range of numbers to examine and how many numbers have already been checked. All Primers report that there are two sums being calculated: a sum of numbers to check and a sum of numbers that already have been examined. Based on the new and old report there is a quantity of numbers calculated that have been examined. Then, the time from the last change is calculated. Based on these two values the speed for node i (1) is calculated.

After the Migration Coordinator gets all reports form Coordinators it starts the calculation. During that time all Coordinators wait for messages. When the agent network is being created at the beginning of an experiment, Migration Coordinator creates a list of objects describing nodes. This list is used for migration calculation but also for remembering names, locations in the network and for information if a node has to report back after migration finishing. As the reports arrive the information in the list is being refreshed.

$$speed_i = \frac{currently\_checked\_numbers_i - last\_checked\_numbers_i}{current\_time_i - last\_change\_time_i} \quad (1)$$

$$estimated\_time_i = \frac{number\_of\_numbers\_to\_check_i}{speed_i} \quad (2)$$

$$agent\_value_i = \frac{estimated\_time_i}{number\_of\_agents_i} \quad (3)$$

$$agent\_change_i = \frac{average\_active\_time - estimated\_time_i}{agent\_value_i} \quad (4)$$

$$proposed\_agents_i = number\_of\_agents_i + agent\_change_i \quad (5)$$

$$norm\_proposed\_agents_i = proposed\_agents_i * \frac{current\_agents\_sum}{sum\_of\_proposed\_agents} \quad (6)$$

For each node we have data on the estimated time (2) and the list is sorted beginning with the shortest time till the end of experiment. Then for each node three values are calculated. The agent value for node i (3) is a measure of how much time from the estimated time till the end falls to one agent. The next value is a bit more complicated (4). In this algorithm there is such a value as an average active time. The term active time applies to those nodes only, where migration can take place or in other words, which have the estimated time higher than the node threshold parameter . So the goal is to calculate how many agents on node i should have the time as close to average as possible. The assumption is that an agent (Primer) represents some work to do and if there was a certain number of agents, then the time would be close to average. Having such simple assumption, the number of proposed agents for each node is calculated (5). If for example a node has a time lower than average - then there should be more agents and the agent value change is greater than zero. If not then some agents should migrate from this node. But after calculating the proposed agent number there is a possibility that there should be more or less agents than currently is, so it is necessary to correct this number on each node by sum of agents divided by sum of proposed agents (6).

After this process agents are distributed according the resources (agents) available in the system. But there is a possibility that still the sums of agents and proposed agents are not equal, so there is correction algorithm, that makes these sums equal by adding or subtracting proposed agents for each nodes starting from those that have the biggest number of proposed agents.After executing this algorithm a list of proposed migrations is created. Building this list is based on making equal the number of agents and proposed agents on a node (in the node information list) possessed by Migration Coordinator. Agents always migrate from the node that has the biggest estimated time to the node that has the lowest estimated time.

5.5 Performance Measurements

The experiment is conducted in order to test the implemented system using big number of computers. Because the College environment is homogeneous, there was a heterogeneity introduced by running more instances on one computer. Tests were conducted on 15 computers in which on five of them there was one JADE container running. On the next five there were two containers running and on the rest 3 containers. The number of containers equals 30. In this experiment the main goal is to find the optimal parameters for the described configuration. Without migration the experiment took 1000.5 seconds and the first computer reported after 317.6 seconds.

5.6 Discussion of Results

The goal is to find optimal configuration for two different environments: home and university. In order to do this there has been a set of experiments conducted. Other main purpose is to show, that migration helps to improve efficiency of task completion when a network is composed of computers, which are not homogeneous - they have different configuration or/and different

computing power at the moment (they can be busy because of other programs running

We spent countless hours making multiple runs in order to assure ourselves that the results were due to inherent randomness and not model errors. Table 1 provides the best values from these runs. Note that the agent migration increases the efficiency in task execution.

Parameter	Heterogeneous computers	Homogeneous computers
Profit in execution time	58%	35.4%
Primes range search unit	150	150-200
Report time	20s	30s
Agents on a node	5-20	10-15

TABLE 1: Optimal configurations for heterogeneous and homogeneous computers

Moreover, we can combine the results of all of above cases and draw the following conclusions. The more agents in the system, the more dynamic the environment is and also the more migrations take place. The most important parameter in the system is primes range search unit, because it is able to balance the calculations efficiency and the communication velocity. The optimal configuration is when all computers finish their tasks in time close to each other. The closer the migration phases are in time, the more migrations there are in the system. To make the system more stable migration phases have to be distant in time. This also impacts the number of messages in the system - the more stable the system is the lower communication costs. The lower number of migrations the shorter the execution time assuming there is enough agents in the system, that are able to cover differences in computer efficiency (the optimal number of agents seems to be around 10 or 15).

When there are migrations in the system, there is a possibility of cycle migration phenomenon. This has a negative impact on nodes efficiency and it was proven using full experiments report. The cause of this lies in the accuracy of time till end estimation for a node and the number of agents. There are two ways of limiting it: low number of agents or limited migration, but there is always a danger that after a sudden event in the system (like one node efficiency collapse) it could not handle changes in a short time (slightly higher execution time).

When the number of agents in the system is small, a limited migration could function more efficiently. The overall conclusion here is that within a dynamic environment the key is to find a balance between covering differences in computer efficiency and unexpected events (the more agents, the more accurate it is) and limiting migrations and migration cycles (the less agents the better).

6. CONCLUSION & FUTURE WORK

The performance results show that a lot of work must be done in the transport area, defining fast Message Transport Protocols, and using lightweight content languages in order to make ACL based migration more competitive in terms of performance.

The upgrades to DDS framework could be connected with two key points: algorithm for migration and time estimation. More advanced algorithm for migration calculation could be more concentrated on avoiding migration cycles (something like limited migration) with parameters regarding how the node was handling the task previously. The time till end estimation could be more influenced by historical efficiency based on assumption that it does not change so often

The DSS proposed has attempted to introduce dynamically distributed service as inherent to an inter-network as FTP, TELNET, e-mail, chat etc. The rationale for - and the main features of the

basic scheme have been described. Mechanisms of DDS are useful because agents can be organized into efficient configurations without imposing external centralized controls. Operation without a central coordinator eliminates possible bottlenecks in terms of scalability and reliability. Process intensive applications will be the main beneficiaries of the scheme.

7. REFERENCES

[1] FIPA. Foundation for Intelligent Physical Agents, http://www.fipa.org

[2] AgentCities.NET. European Commission funded 5th Framework IST project. November 2001. http://www.agentcities.net

[3] JADE, Java Agent DEvelopment Framework, http://jade.tilab.com

[4] F.Bellifemine, G.Caire, T.Trucco, G.Rimassa. JADE Programmers Guide, July 2002.

[5] Hmida, F.B., Chaari, W.L., Tagina, M.: Performance Evaluation of Multiagent Systems: Communication Criterion, In Proc. KES-AMSTA 2008, LNAI 4953, 2008, 773-782.

[6] Bernon, C., Chevrier, V., Hilaire, V., Marrow, P.: Applications of Self-Organising Multi-Agent Systems: An Initial Framework for Comparison, Informatica,30,2006, 73-82.