International Journal of Multidisciplinary Approach and Studies

ISSN NO:: 2348 – 537X

Implementation of the Mobile Agent System using Resource Allocation Problem

Sudan Jha

PhD Scholar, PG Department of Computer Science and Applications,Utkal University, Vani Vihar

&

Associate Professor, Principal, Eastern College of Engineering, Biratnagar, Nepal

ABSTRACT:

A mobile agent is a program that autonomously migrates from machine to machine in a heterogeneous network, interacting with services at each machine to perform some desired task on behalf of a user. As a mobile agent migrates from machine to machine in a heterogeneous network the environment in which it operates changes. The ability to adapt to dynamic environment and unexpected events is a key issue for mobile agents. In this one, we first present a model of adaptive mobile agents. We then discuss the implementation of a task execution system based on adaptive mobile agents. On each machine it interacts with service agents and other resources to accomplish its task, returning to its home site with a final result when that task is finished. Characteristics of mobile agents include agent mobility, object passing, autonomous, asynchronous and parallel Execution. The main reasons to use mobile agents in distributed applications are they reduce network load, execute asynchronously, autonomously, adapt dynamically and they are naturally heterogeneous. Mobile Agent System provides the features of agent mobility, creation and termination of agents, agent cloning & getting the results of completed tasks from the other agents.

Keywords—Mobile Agent, Migration; Object Passing; Mobile Cloning; Mobile Tasks**;**

I. INTRODUCTION

A mobile agent is a program that autonomously migrates from machine to machine in a heterogeneous network, interacting with services at each machine to perform some desired task on behalf of a user. As a mobile agent migrates from machine to machine in a heterogeneous network the environment in which it operates changes. The ability to adapt to dynamic environment and unexpected events is a key issue for mobile agents. In this one, we first present a model of adaptive mobile agents. We then discuss the implementation of a task execution system based on adaptive mobile agents.

On each machine it interacts with service agents and other resources to accomplish its task, returning to its home site with a final result when that task is finished. Characteristics of mobile agents include agent mobility, object passing, autonomous, asynchronous and parallel Execution. The main reasons to use mobile agents in distributed applications are they reduce network load, execute asynchronously, autonomously, adapt dynamically and they are naturally heterogeneous. Mobile Agent System provides the features of agent mobility,

creation and termination of agents, agent cloning & getting the results of completed tasks from the other agents.

A distributed application can be viewed as a collection of mobile agents that independently move around in a network and communicate with each other to achieve some predefined goals. Implementing distributed applications using mobile agents can have several advantages:

Bandwidth Conservation: - In scenarios where a client is connected by a low bandwidth link to one or more servers from which it needs to download and then process a large amount of data, it is advantageous to send the processing code in a mobile agent to the servers rather than download the large volume of data over a low bandwidth link. The processing code can operate on the data at the servers and then just send the final results back to the client.

a. *Better Resource Utilization:* - While an agent is executing at a particular server other agents can execute parallel tasks at remaining servers keeping resources busy always.

b. *Load Balancing:* - Mobile agent based systems provide a mechanism for load balancing. Agents can clone themselves to share the load and can migrate from an overloaded machine to an under loaded machine autonomously.

Since a mobile agent roams from machine to machine in order to achieve its objectives, the environment in which it has to execute may change considerably. Examples of such changing environment parameters can be change in CPU load, available network bandwidth, availability of other required resources etc. The environment is also dynamic and changes with time. For example, the CPU load of a machine can change while an agent is executing there because of arrival of other jobs from outside or the agent may move to a part of the network with low network bandwidth. In addition to changing environments, a mobile agent may also have to react effectively to unexpected events.

In order to work in changing environments and react to unexpected events, a mobile agent may need to adapt to the current environment in order to achieve its goals correctly and/or efficiently. In this one, we first present a model of an adaptive mobile agent system, the model species the components of an adaptive mobile agent and their behaviors with respect to the current environment. We then describe the design of an adaptive mobile agent based task scheduling system. Adaptive mobile agent system provides the features of agent migration, tracking of a mobile agent and Inter-Agent Communication.

II. DESIGN OF THE ADAPTIVE MOBILE AGENT MODEL

An agent can be viewed as satisfying an ordered set of goals to achieve some overall objective. The agent takes a sequence of actions in order to satisfy the next goal in the set.

Adaptation can be viewed as changing the goal set. The effect of the change can be a new set of actions to achieve the same overall objective as before.

1. ***Adaptive Mobile Agent Components***: - The adaptive mobile agents internal functionality consists of two components, a Mechanism and an Adapter
2. ***Mechanism:*** - The Mechanism is the only interface of the mobile agent to its environment, and it contains sensors to assist it in this purpose. The sensors periodically sense the environment and report their findings to the Mechanism, which then analyses these sensor inputs. The Mechanism then creates a view of the current environment using the sensor inputs. This view of the environment as created by the Mechanism is called Percept. The detectors carry out the commands issued by the mechanism, and thereby affect the environment.

Fig.1 shows the basic structure of an adaptive mobile agent and their interactions.

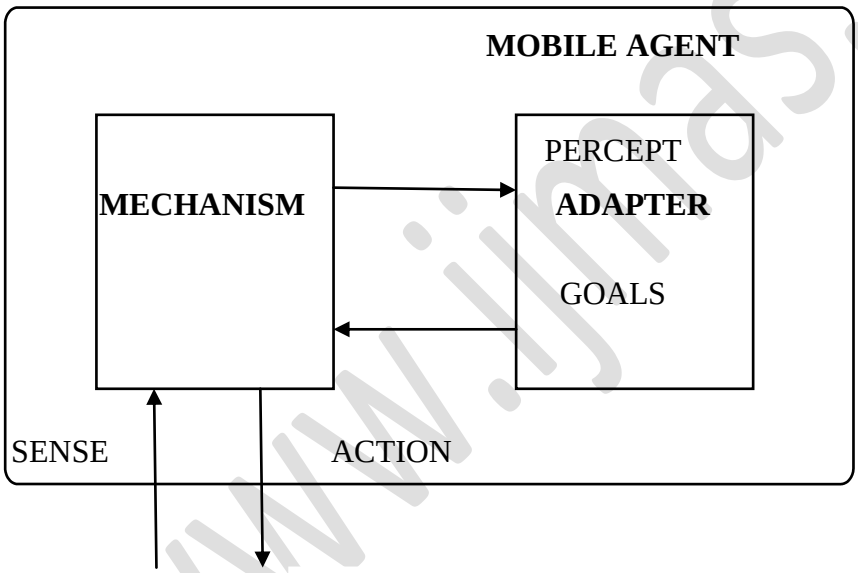

Figure 1: Architecture of Proposed Detection

***2.1 Adapter*:** - The Adapter is the component, which tells the mobile agent whether adaptation is needed and if so then how to adapt. In our model, the Adapter has to search through all the various adaptation possibilities of and the best way to adapt. Only after the Adapter has exhaustively searched through all the adaptation possibilities, can it answer the question whether there is any need to adapt at all. In this proposed model, every adaptation possibility maps a set of goals before adaptation to a set of goals after adaptation; thus, changing the set of goals in our model does adaptation.

***2.2 Behavior States of the Mobile Agent*:** - Fig.2 shows the various behavioral states of an agent. An agent can be in any one of two states: Mechanism state and adapter state. The

Mechanism can be in extract Goal, execute command, evolutionary behavioral states and Adapter can be in only adapt behavioral state.

2.3 Mechanisms extract Goal behavior: - When the behavior attribute in Mechanism's state has the value extract Goal, it signifies that the Mechanism is currently generating commands for the detectors to carry out. In this behavioral state, the Mechanism picks up the current goal to be executed from the ordered set of goals it has to fulfill, and then generates commands to execute it.

***2.4 Mechanisms execute Command behavior*:** - In this behavioral state, the Mechanism acts out the commands issued in the extract Goal behavioral state. The detectors carry out the commands and thereby detect the environment and agent's state.

***2.5 Mechanisms eval behavior*:** - In this behavioral state, the Mechanism senses the environment and forms a view of the current environment. An interrupt like timeout, coded adaptation, etc., can change the behavioral state from extract Goal or execute Command to *eval*.

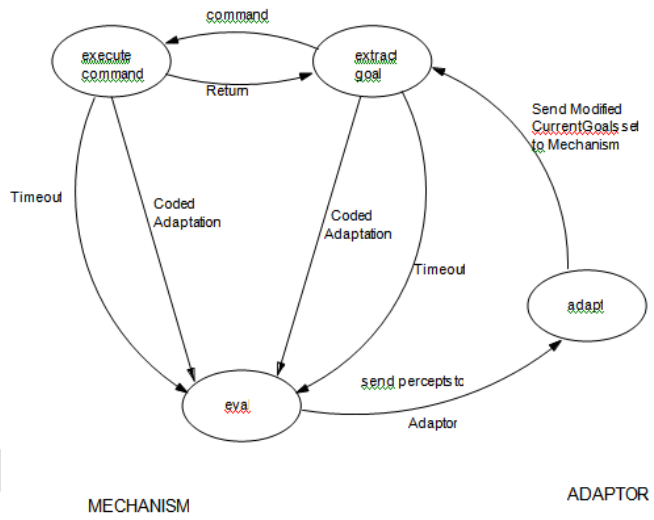

Figure 2: Agent Sate Diagram

***2.6 Adapters adapt behavior*:** - As soon as the Mechanism has sensed the environment, it sends the percepts collected to the Adapter. When this transfer is over the Adapter's behavior state is set to adapt. The behavioral state of Adapter being set to adapt, just signifies that the Adapter is now activated. When the Adapter has decided on the adaptation possibility to be adopted, and hence modified the set of goals, the nit sends the modified current set of goals to the Mechanism. When this transfer is over, the control returns to the Mechanism and its behavioral state is set to extract Goal. Now, the mobile agent begins a new round of normal execution, where it continually picks up the immediate goal and acts on it.

III. DETAILS OF MECHANISM AND ADAPTER

***3.1 Mechanism*: -** Mechanism is the part of the Mobile Agent, which contains its sensors and detectors and hence provides the means for interacting with the environment. The sensors sense the environment as the name suggests. The sensors can be used in the following cases,

i. When the Mechanism is in either the extract Goal or execute Command states, it may make use of the sensors. This use of sensors is particularly hard coded in the mobile agent code.

ii. When the Mechanism is in the *eval* behavioral state. This means that when the mobile agent is interrupted from its normal execution and hence has to adapt, it first moves to the *eval* behavioral state and then senses the environment. Thus, when an interrupt is generated and sent to the Mechanism, it changes its behavior to the *eval* state. These interrupts can be generated in the following cases,[2]

a. On timeout by the Timer. Timer can be thought to be a part of Mechanism.[2]
b. Coded Adaptation i.e. when the need to check for adaptation is coded in the mobile agent program.

The detectors act on the commands issued by the Mechanism, and thus affect the environment as well as the agent's state. Depending on the current set of goals and the percept from the environment (if any), the Mechanism then issues some commands to the detectors. The execution of these commands forms the action on the environment.

***3.2 Adapter*: -** Adapter does adaptation whenever agent asks for adaptation. It has to find out whether there is any need to adapt or not and if there is a need, find the best way to adapt.

***3.3 Adaptation*: -** Adaptation is the behavior of an agent in response to unexpected (low probability) events or dynamic environments. We model adaptation by changing the current set of goals of the Mobile agent to a new set of goals which has a higher probability of success in achieving its final or long term goal(s) under the current environment.

IV. RESOURCE ALLOCATION PROBLEM

***4.1 The Problem*: -** The Resource Allocation problem involves completing a group of tasks. The input to the problem is a task structure, which shows the input-output dependencies between the different tasks. A typical task structure is shown in Fig.3. Each individual task has some resource requirements, all of which have to be satisfied before the task can proceed. As can be seen in Fig.3, task T1 has the resource requirements r1,r2,r3. A deadline and expected duration for a task are also specified. As can be seen in the Fig.3 the deadline for the task T1 is 10,000 milliseconds since the time when the execution of the task structure is started. The expected duration for the task T1 is 3000 milliseconds i.e., the task T1 will execute and hence use the CPU for approximately 3000 milliseconds.

This problem is intended to be solved using a system of mobile agents. After the user specified the task structure, it is traversed using ADFS traversal algorithm.

At every task node (current task), the algorithm takes a look at the set of tasks which depend on current task's result directly i.e. those tasks to which the current task sends out an edge. These tasks are parallel is able and hence can be solved using different agents. Thus, a goal of cloning <**number of paralleisable tasks - 1**> agents is assigned to the agent which had the responsibility for the current task.

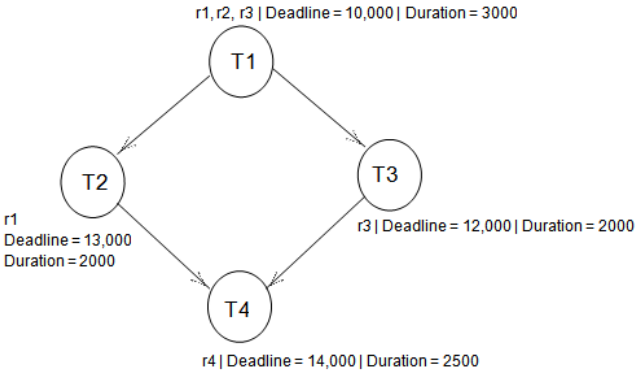

Figure 3: The Resource Allocation Problem

V. IMPLEMENTATION

5.1. Goals: - With reference to Fig.3, initially only one mobile agent is launched with an initial set of goals. It also carries with itself the set of goals for the agents it has to clone. After cloning, it assigns to the clone its corresponding set of goals. The rest of this chapter describes what happens during the execution of this task structure with reference to the theoretical model described earlier.

The set of goals generated for the task structure shown in Fig.3, have been listed in

Appendix A

Here only a modified subset of the set of goals is used for illustration of the theoretical model.

Current Goals:

(1) MAgent 0.

(2) Schedule Task T1 before $T_{1deadline}$ ¡ $T_{1duration}$

(3) Coded Adaptation

(4) Work at Server N1

(5) Wait for resources r1, r2, r3 for a maximum of f($T_{1deadline}$) where task duration is $T_{1duration}$

(6) Complete Task T1

5.2 Working of the Agent System for this Problem: -

***5.3 Selection of a Server suitable for Task T1*: -**

Perception by Mechanism: - Say, there are Servers N1, N2 and N3 in our system with resources (r1), (r1,r2,r3), (r1,r2,r3,r4). Then, the initial environment is:

ENVIRONMENT = f

Servers = (N1, N2, N3)

N1.OwnedResources = (r1) ,

N2.OwnedResources = (r1, r2, r3) ,

N3.OwnedResources = (r1, r2, r3, r4) g

If the server from which the mobile agent MAgent0 is launched say Server N1, then the initial agent state for MAgent0 is AGENT STATE = < location = N1, behavior = extract Goal, Time = 0 > <d>.

5.4 Example: - Let's consider the moment when the agent MAgent0 has to execute the goal3. In the extract Goal behavioral state, the Mechanism extracts the goal 3, and issues the commands corresponding to this goal

COMMAND =f generate Coded Adaptation interrupt g

Then, the behavioral state of the Mechanism is changed to execute Command

In the execute Command behavioral state, the Mechanism acts out the command issued in the previous behavioral state, whereby it generates a Coded Adaptation interrupt. The result of this interrupt is that the behavioral state of the Mechanism is changed to eval. In the eval behavioral state, the Mechanism senses the environment and sends the corresponding Percept to the Adapter. The Sensor Inputs to the Mechanism form the percept,

PERCEPT = (f {N1.Resources})

Owned =(r1), Expected Migration Time from N1 to N2 = migrationTime1,Expected MigrationTime from N1toN3 = migrationTime2 g.

***5.5 Adaptation*: -** This percept is sent to the Adapter, which then modifies the next goal, goal 4. Since the required resources for the task T1 are (r1,r2,r3) which are not present on N1, so the Adapter decides to migrate to either N2 or N3. Now suppose, that migrationTime1<migrationTime2, so there is a higher possibility of satisfying the motivation

by working at ServerN2, as it will take less time to move onto N2. Thus, the goal 4 is modified to Work at Server N2. Then, the Adapter sends the modified set of current goals to the Mechanism.

***5.6 Action by Mechanism*:** - Once, the Mechanism has got the set of modified goals, it extracts the next goal, goal 4 in the extract Goal behavioral state . Then, it issues the command Migrate to Server N2 corresponding to goal 4. Then, the behavioral state of Mechanism changes to execute Command. In this behavioral state, the Mechanism's detector's execute the command, whereby the location of the mobile agent is changed to N2.

***5.7 Adaptation due to resource not free*:** - Perception by Mechanism Say, MAgent0 acquires the resource r1 but is not able to acquire the resource r2. Right now the control of execution is with the Mechanism which is in the execute Command behavioral state. MAgent0 now feels that this is a suitable time for making an adaptation decision. So ,the behavioral state of MAgent0 is changed to state in which it senses the environment. The sensor inputs tell the Mechanism that the expected time that MAgent0 may have to wait before it is able to acquire all the resources at N2 is waitTime1. At the same time the expected migration time from N2 to N3 is migrationTime1. The Sensor Inputs to the Mechanism form the percept, PERCEPT=f N2.ResourcesOwned=(r1, r2,r3), N2.waitTime = waitTime1, migration Time(N2, N3) = migrationTime1 g.

VI. IMPLEMENTATION OF THE MOBILE AGENT SYSTEM

Mobile agent system consists of Agent Servers running in the network, simulation launchpad for specifying simulation parameters, Task graph GUI for giving task graph input and an Inter Mobile Agent Communication (I.M.A.C.) Server running in the network, as well.

1. How the system works
2. The user input
3. Where should the agent migrate?
4. Getting the results of other tasks
5. Adaptation

6.1 Functional Blocks: - The implementation of the whole system can be divided into four main functional blocks.

i. **Agent Host** -provides the functions needed for agent migration.

ii. **Launcher** -provides functions for giving task graph as input, finding number of agents required for completing the job, and arbitration of which agent does which task, goals generation for each agent, Launching the parent agents for this job.

iii. **Communication** - provides the functions for getting the results of the completed tasks, the view of the system to the user i.e., where the agents are executing, status of the agents (migrating or adaptation or completed).

iv. **Simulator** - provides the simulation environment for the system.

6.2 Agent Host: - This functional block provides the functionalities needed for migration of agents. The mobile agent platform consists of a number of agent servers. All those machines which desire to host/launch mobile agents should have the agentHost.server running. When an agent migrates from one agent server to other, it carries with itself the code on which it executes and the current status of its variables. In our system developed in Java, executable code consists of the .class files that the agent uses. The current status of all the variables used by the agent is encapsulated in a Java object.

6.3 Object Migration: - Any agent programmed in our platform should implement the java.io. Serializable interface which allows an agent object to capture the current status of its variables in a standard format. If an agent class declares some other non-simple classes as instance variables, then the values of the entire instance variables of such classes are also stored. The only requirement is that all such classes which are used as instance variables of the agent class should also implement the java.io. Serializable interface. This representation of an agent object is then transferred from one agent server to the other using normal client – server sockets.

***6.4 Code Migration*: -**It is the responsibility of the mobile agent to provide a list of all the .classfiles that it might need at a particular agent server. These files are then transferred using client - server sockets. When a agent requests the agent server where it is currently staying to migrate to other agent server , the current agent server then establishes a socket connection with the destination agent server and transfers the agent code and object. After this transfer is over, the current agent server stops the agent thread that had requested to be migrated and the destination agent server starts the agent thread on receiving the agent code and object.

6.5 Launcher: - This functional block takes a task graph as input from the user, and then it decides how the tasks in the graph are to be completed and starts the initial agents. It defines the number of agents required to complete the job, goal generation for each agent, Parallelization of tasks and launching the parent agents for this job. Task graph input consists of Taskname, list of resources that a task needs, expected CPU usage by the task, Input - output dependencies. After the Task Graph has been completely specified by the user a "Depth First Traversal" is done to find the set of tasks, which are paralleisable at any instant. At every point of traversal a goal string is generated and assigned to an agent.

***6.6 Goal Generation Algorithm*: -**

Step 1: Find the tasks which do not depend on the results of any other tasks in the task graph. These tasks form the initial set of paralleisable tasks. Generate the goal string \Make <number of paralleisable tasks> \ Clones ". This goal string is executed by the launcher, which makes as many agents as mentioned in the goal string and launches them. Assign each such paralleisable task to an agent.

Step 2: Start a Depth First Search Traversal from each such paralleisable task found in Step 1.

Step 2a: Find the tasks on whose results the current task depends on Assignthegoalstring\Waitfortheresultsofthefollowingtasks..." to the current agent.

Step 2b : Find the tasks which are reachable from the current task. Henceforth the agent assigned to the current task would be referred to as the current agent.

Step 2c : if the number of tasks found in Step 2b>1, then these tasks form the set of paralleisable tasks at this instant.

then

Generate the goal string

\Make <number of parallelizable tasks - 1> Clones "; Assign the responsibility of cloning to the current agent; Assign one of the paralleisable tasks to the current agent; Assign one from the rest of the paralleisable tasks to one of the clone agents

else if the number of tasks found in step 2b = 1

then

Assign the task to the current agent.

else if the number of tasks found in step 2b = 0

then

Assign the goal string \Pause" to the current agent.

Step 2d : Pick up one of the tasks found in step 2b and continue the dfs traversal from there.

After the set of goals have been generated for the task graph, the launcher executes the first goal string. The first goal string tells the launcher the number of agents it has to make. The launcher then makes an agent object, assigns to it the set of goals that it has to execute, and launches it. After being launched, an agent executes each of its goals one by one. The agent is free to roam on the net and can visit any of the machines which have the agent server running on them. A task-agent's code consists of a goal interpreter which associates commands with every goal string. The Mechanism extracts a goal one by one and issues the commands associated with it in its extract Goal behavioral state. Then the commands are acted upon by the Mechanism in its execute Command behavioral state.

***6.7 Communication*: -** This functional block provides the user a view of the system at any time instant. It also serves as a centralized database for the agents present in the network and there by acts as runtime communication between agents. It maintains the following information-the agents present in the network currently, the current location of an agent,the current task being executed by an agent, the status of the task currently being executed, the results of the tasks it has completed, History of an agent - the tasks it has completed, their results, the locations it has visited. All the agents in the system know the location of the I.M.A.C. server and hence keep sending their current location, current task,etc.,to it frequently. Other agents may use this database if they so desire during their execution.

***6.8 Simulator*:-** This functional block provides the simulation environment for the system. The motivation behind simulating an environment is to be able to study the behavior of the system when some changes are simulated in the environment. The simulator consists of two distinct parts, Simulation Launchpad and Simulator Agents.

***6.9 Simulator Agents*:** - Once a simulator agent has been launched by the Simulation Launchpad, it migrates to the machine it has to simulate on. The simulation of a task involves the following steps,

²Generation of task: The simulator randomly decides the number of resources that the task would require and then randomly selects these many resources from the list of resources it can simulate.

²Deciding the task arrival time: The load parameter is mapped to an inter-task arrival time range. The simulator agent picks up an inter-task arrival time from this range. The simulator agent then waits for this much time after having scheduled the last task, before it picks out the current task from the ready queue and schedules it.

²Random generation of the CPU Usage time for the current task

For each resource required by the current task, the simulator agent picks up a random value from the resource usage time range specified by the user for this resource type. Then, the CPU Usage time for the task is calculated as a mean of the random values picked for each of the required resource.

²Interface for acquiring the resources and using them. Both the simulator agent and the task agents make await Or Acquirethread to acquire the resources needed and then use them. The waitOrAcquire thread keeps on trying to acquire the resources till either all the resources are acquired, or a timeout occurs. Thus, any entity, which wishes to acquire a set of resources and then use them, must specify the following parameters, the set of resources to be acquired, a timeout period within which the waitOrAcquire thread should try to acquire the resources, the usage time for the resources.

VII. CONCLUSION AND SCOPE FOR FUTURE WORK

In this paper a theoretical model for adaptive mobile agent system has been developed. A mobile agent platform providing the features of Migration, Tracking, Collaboration among agents and Adaptation has been developed. The model for adaptation has been tested using a Resource Allocation Problem. The behavior of the system under different load conditions, different migration costs, etc., can be studied as a part of continuation of this work. The most exciting phase in any such continuation should be when instead of using a simulator to simulate the environment for the agent system, a real network environment is used. One proposal for any such experimentation is that different probability distributions can be used to simulate the arrival of tasks in the simulator executing on every agent server. Then a study can be made to find which probability distribution follows the real network conditions closely. Another important future work can be to change the design of the simulator. Presently, the simulator has been made static i.e., no changes can be made to the simulation

parameters once the simulator agents start their simulation. A proposal is to be made the simulator dynamic so that the values of parameters can be changed during run-time. This will provide more control while studying the new adaptation policies. For example, if we are able to some how make are source Busy just when a task agent arrives on that server looking for that particular resource, then we would be able to study the adaptation policy more efficiently.

REFERENCES

i S. Ranjan, A. Gupta, A. Basu, A. Meka and A. Chaturvedi, \Adaptive Mobile Agents: Modeling and a Case Study", Workshop on Distributed Computing (WDC), Calcutta, India.

ii Keith Decker, Katia Sycara and Mike Williamson, \Intelligent Adaptive Information Agents", AAAI'96 Workshop on Intelligent Adaptive Agents, 1996.

iii Richard Goodwin, \Formalizing properties of Agents", September 12,1994.

iv Robert Gray, David kotz, George Cybenko, Daniela Rus, \Mobile agents:

v Motivations and state-of-the-art systems", Technical Report TR-2000-365, Department of Computer Science, Dartmouth College.

vi Kunda.Z.,\The case for motivated reasoning", Psychological Bulletin, 108 (3), pp. 480-498.

vii M. Luck and M .d'Inverno, \A formal framework for agency and autonomy", In proceedings of the First International Conference on Multi-Agent Systems, pages 254-260. AAAI Press MIT Press, 1995.

viii Onn Shehory, Katia Sycara, Prasad Chalasani and Somesh Jha,\Agent Cloning: An approach to Agent Mobility and Resource Allocation", IEEE Communications, 36 (7), pp. 58-67.

ix Michael Wooldridge, Nicholas R. Jennings, \Intelligent Agents: Theory and Practice", Knowledge Engineering Review, 10 (2), 1995.

x Mark d'Inverno, Michael Luck, \Development and Application of a Formal Agent Framework".

xi Daniela Rus, Robert Gray and David Kotz, \Autonomous and Adaptive Agents that Gather Information".

xii Antonio Corradi, Rebecca Montanari, Gianluca Tonti and Cesare Stefanelli, \How to Support Adaptive Mobile Application"

xiii www.1000projects.org