Adaptive Mobile Agents: Modeling and a Case Study

Supranamaya Ranjan
Department of Electrical & Computer Engineering
Rice University
Houston, Texas - 77005

Arobinda Gupta **Anupam Basu** **Anand Meka** **Abhishek Chaturvedi**

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur – 721302

ABSTRACT

Mobile agents have been proposed as a novel and useful paradigm for designing distributed applications. Mobile agent based distributed applications are specially suited for mobile computing environments involving different types of devices because of better bandwidth conservation, support for disconnected operations, easier device/user-specific customization etc. As a mobile agent migrates from machine to machine in a heterogeneous network, the environment in which it operates changes and it may encounter unexpected events like faults etc. The ability to adapt to dynamic environment and unexpected events is a key issue for mobile agents. In this paper, we first present a model of adaptive mobile agents. We then discuss the implementation of a task execution system based on adaptive mobile agents and present results to show that adaptation can be very useful for mobile agent based distributed applications.

Keywords: Mobile, Agent, Distributed, Environment, Adaptation

1 INTRODUCTION

Mobile agents have been proposed as a novel and useful paradigm for designing distributed applications. 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. An agent can, under its own control, suspend its execution, migrate to a new machine, and continue execution at the new machine from the point it left off. 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 [3]:

- **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.
- **Support for disconnected operations:** Conventional distributed systems communicating with RPC like mechanisms require a connection to be maintained during a client server interaction. However, in mobile computing applications, a mobile client may not have a continuous connection because of noisy links or power conservation reasons. In such scenarios, mobile agents can be launched from the client to perform the desired services and the user can disconnect while the services are being performed. The results can be sent back to the user when reconnection occurs.
- **Better customization:** A server may want to provide customized services to a client based on the capabilities of the client (for ex., different device types) or user profiles. An agent with the customization code can be supplied by the client itself which can move to the server to customize the service for the client. The server just needs to provide an interface to the services.
- **Dynamic deployments:** Software services can be easily deployed by administrators dynamically by launching mobile code from a central system that migrate and reside at different servers, rather than installing copies of the software at all servers individually.
- **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 underloaded machine autonomously.

The number of different types of devices that are connecting to the web is growing rapidly and it is expected that this number will grow even more in future with a myriad of household and handheld devices coming into the market. These devices are expected to have low computing and storage capabilities, and will connect to a host of services through low bandwidth wireless links. Given the heterogeneous devices and their capabilities, mobile agents will be a strong contender as a general framework for designing distributed applications because of the above advantages. Mobile agents have already been used to build many distributed applications by various research groups, including information retrieval, collaborative work [9], resource allocation [8], network routing [6], brokerage applications, auction sites etc.

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, fault conditions, 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. For example, consider an application in which users launch mobile agents from handheld devices to search for items meeting certain criteria in different auction sites. The agents inform the user if any such item is found. The user then bids for the item of his choice. A mobile agent, while visiting the auction sites, may find an item that is highly suited to the user's needs and is being bid aggressively by agents of other users. The link back to the user may be temporarily down and trying to inform the user may take a long time, by which time the item may be gone.

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. As an example, consider a mobile agent launched to search and retrieve images matching a particular criteria from image databases on the web. Under normal circumstances, the agent will download the images found to the user. However, if the agent determines that the network path back to the client is noisy and poor at this point of time, it may decide to send only a low resolution image (if available) to the user now, leaving the full retrieval for later. As another example of adaptation, in the auction-bidding example above, the agent may autonomously decide (based on various possible factors including degree of autonomy granted by user, bidding history and financial status of the user etc.) to bid for the item anyway on behalf of the user without informing the user first. The ability to adapt to changing environments and unexpected events will be a key factor in the design of mobile agent based systems.

In this paper, we first present a model of an adaptive mobile agent. The model specifies the components of an adaptive mobile agent and their behaviors with respect to the current environment. The model is based on earlier work by Goodwin [2] on modeling robotic agents and on work by Luck et. al. [5] on modeling autonomy of an agent, and our work borrows some concepts and terminology from them. We then describe the design of an adaptive mobile agent based task scheduling system that we are currently implementing. The task scheduling system shows the feasibility and advantages of integrating adaptation into mobile agents. The organization of the rest of this paper is as follows. Section 2 introduces the model of an adaptive mobile agent. Section 3 describes the details of the task scheduling system. Section 4 discusses related works, Finally, section 5 contains some concluding remarks and scope for future work.

2 MODELING AN ADAPTIVE MOBILE AGENT

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, or it may even result in a new overall objective if the original objective cannot be achieved anymore in the current environment.

In our model, a mobile agent consists of two components, a *Mechanism* and an *Adapter.* The Mechanism is the interface of the mobile agent to the environment. The Mechanism contains sensors that periodically sense the environment parameters and report their findings to the Mechanism. It also contains effectors that can take actions to change the environment the agent is in. The Adapter is the component that decides whether adaptation is necessary and if yes, how best to adapt to the current environment. The Mechanism senses the environment through the sensors, analyses them, and creates a view of the environment called a percept. The percept is passed on to the Adapter, which uses it to decide whether adaptation is necessary or not. If adaptation is needed, a new set of goals is passed on to the mechanism, which then transforms the set of goals into a set of actions to be carried out, and then carries out the actions. The effectors are used to make any environment change specified in an action. Figure 1 shows the basic structure of an adaptive mobile agent and their interactions. We next discuss each component in

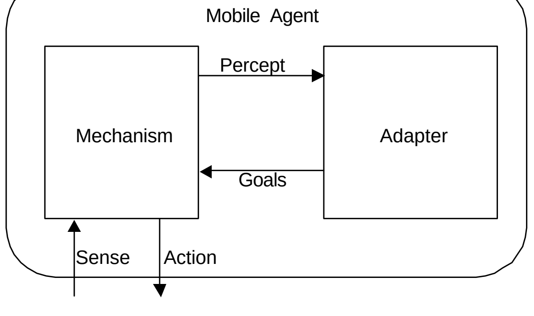

Figure 1. Components of a mobile agent

more details.

2.1 THE MECHANISM

The state of the Mechanism can be specified by a 3-tuple *<S, L, T>*. S is the behavioral state of the Mechanism which identifies what the Mechanism is doing currently, L is the current location of the agent, and T is the duration the Mechanism has spent in its current state. The state variable S can take one of three possible values: extractGoal, executeCommand, and eval. In the extractGoal behavioral state, the Mechanism picks up the current goal to be executed, and generates the set of commands for it. In the executeCommand behavioral state, the generated commands are carried out. In the eval behavioral state, the Mechanism senses the environment and forms a current view of the environment to be passed on to the Adapter.

Figure 2 (shown at the end of the paper) shows the different states the Mechanism can be in and the possible state transitions. The Mechanism normally rotates between the Normal_1=<extractGoal, L, T> and Normal_2=<executeCommand, L, T> states, where L contains the current location of the agent, and T is reset to 0 every time a state transition occurs. The state Normal_1 is entered initially on receiving an ordered set of goals from the adapter. The commands for the next goal in the ordered list is generated, and a transition to state Normal_2 occurs. In state Normal_2, the commands are executed, and transition occurs back to state Normal_1 in order to generate the commands for the next goal in the list of goals. The process continues until all the goals in the list are executed. However, the Mechanism may go to the Eval_Env =<eval, L, T> state from any of the Normal_1 or Normal_2 states if any of the following happens: a timeout, a fault, or an explicit command in the application code itself. We will refer to the third cause as Coded Adaptation, since the application code asks for the environment to be sensed explicitly possibly for adaptation reasons. A timeout can happen if an action is not carried out within a specified time in the Normal_2 state, which can be detected by the T component of the Mechanism state. This may indicate changed environment and may force the agent to sense the environment and reevaluate if adaptation is necessary. In the Eval_Env state, the environment is sensed and a percept is sent to the Adapter to see if any adaptation is necessary. Note that the environment can also be sensed in the Normal_1 and Normal_2 states if necessary, but in those states, the values sensed are used internally and no interaction with the Adapter occurs.

2.2 THE ADAPTER

The Adapter state consists of a tuple <S, T>, where S is the behavioral state which can only take the value adapt, and T is the time spent in the adapt state. Thus the state diagram of the Adapter is very simple and is shown in Figure 2.

In order to describe how the adaptation process works, we first need a few definitions. An attribute is a perceivable feature of the environment. A percept is a set of attributes. Thus, a percept is nothing but a view of the environment. An adaptation method is a single mapping from a percept to a set of goals. An adaptation policy is a set of adaptation methods. Thus, the adaptation policy specifies the possible ways in which the mobile agent adapts to different environments.

Given an environment, there may be different ways an agent can adapt. Thus some type of ranking of the adaptation methods in the adaptation policy is necessary. This is achieved by a motivation degree function. Motivation was defined by Kunda [4] as “any desire or preference that that can lead to the generation and adoption of goals and which affects the outcome of the reasoning or behavioral task intended to satisfy the goal”. We associate with each adaptation method a motivation degree, which is the probability of success in achieving the final goal if the set of goals corresponding to the adaptation method is selected as the current set of goals. The Adapter then selects the adaptation method with the highest motivation degree corresponding to the current environment. Note that the set of adaptation methods (the adaptation policy) and the motivation degree funcion can be hardcoded by the user or can be learnt dynamically from history. Usually it will be a combination of both where the user specifies an adaptation policy and a motivation degree function, which then can be modified dynamically as well.

Thus, on receiving a percept from the Mechanism, the Adapter goes through the set of adaptation methods, looking for the ones that match the percept. The one with the highest motivation degree is then chosen, and the current set of goals is modified to be the one corresponding to that adaptation method. The new goal set is passed to the Mechanism, which then generates and executes commands for the set of goals. If no adaptation method matches the current environment, adaptation is deemed unnecessary and no change to the goal set occurs. Thus, no adaptation can also be viewed as a special adaptation method.

2.3 THE AGENT STATE

The state of the mobile agent is the 3-tuple <M, A, App>, where M is the Mechanism state, A is the Adaptor state, and App is the Application-specific state for the mobile agent. Thus, an adaptive mobile agent can be completely described at any point of time by its state, the adaptation policy, and the motivation degree function.

3 CASE STUDY: A TASK SCHEDULING SYSTEM

We implemented a simple task execution system based on adaptive mobile agents. The input to the system is a task graph for a job consisting of multiple tasks. Each individual task has some resource requirements, all of which have to be satisfied before the task can be started.

A task also has a given duration for which the task runs. The task graph is to be executed on a heterogeneous network of servers, each of which has a (possibly different) set of resources. The set of servers and the resources they have are static, and this information is available at all servers. However, the load conditions at the servers can vary with time. The aim of the system is to complete all the tasks on the network as fast as possible subject to dependency and resource constraints.

The basic execution process proceeds as follows. A task is executed by a mobile agent. A single mobile agent may execute multiple tasks or may clone itself to handoff some tasks to the cloned agents to be done in parallel. A mobile agent executing a task migrates to a server that has the resources that are required by the task, and tries to acquire all the necessary resources. If the resources are acquired immediately, the task is executed. Otherwise, the lo ad condition of the server is sensed to get an idea about the time the task may have to wait to acquire all resources. The agent relies on history to make this adaptation decision. The agent looks at the average waiting time of tasks with similar resource requirement in the recent past. If the migration time to a remote server (with the required resources), plus the expected waiting time at that server is less than the average waiting time at the current server, the agent decides to migrate to the remote server. If the average waiting time is less, the agent decides to wait and retry to acquire the resources at the current server itself for a duration equal to the expected waiting time. If the resources are still not acquired at the end of this duration, a timeout occurs, forcing the agent to sense the current environment (the expected waiting time) again and make an adaptation decision whether to migrate or not. The process is repeated till all the tasks are completed. The expected waiting time at the current server and at the remote server is provided by an underlying environment sensing system. In our implementation, we have implemented a simulator that simulates this environment sensing system. The migration time between servers is static and fixed a-priori in the current implementation. The details of the implementation are omitted here due to space constraints.

3.2 EXPERIMENTAL RESULTS

We implemented the task execution system following the model we proposed. The agent framework for creation, migration, and cloning of mobile agents is also implemented by us. There are 3 servers, N1, N2, and N3 in our setup, with zero or more copies of three resources r1, r2, and r3. The load conditions of the servers are varied in the simulator. Each server can be underloaded, normally-loaded, or overloaded. For a fixed resource structure and load conditions of the servers, we executed a number of task graphs in the system, varying the migration time between servers for each task graph. We measured the number of migrations and the completion time of the task graph in each case. The tables below show some of the results. The task graphs, along with the resource requirement of each task, are generated randomly.

Table 1 shows the behavior of the system when all required resources for any task in the task graphs are available at all the servers, all three servers are overloaded, and the migration time between servers is very high. As expected, the number of migrations is 0 (since the migration time is very high), all tasks finish at the same server (the server at which the job is launched), and the completion time is high since the server is overloaded. The time is indicative of the cost of executing the job at a single overloaded server.

Task Graph	No. of Tasks	No. of Migrations	Completion Time (sec.)
T1	35	0	232
T2	39	0	430
T3	40	0	281

Table 1. All servers overloaded with same resources and high migration time

Table 2 shows the case when everything is same as in the case in Table 1, except that all the servers are underloaded. The high migration time again prevents any migration to remote server and all tasks are executed at the same server. The time is much lower because the server is underloaded. This time is indicative of the cost of executing the job at a single underloaded server.

Task Graph	No. of Tasks	No. of Migrations	Completion Time (sec.)
T1	35	0	90
T2	39	0	151
T3	40	0	93

Table 2. All servers underloaded with same resources and high migration time

Table 3 (shown at the end of the paper) shows the case when all resources are again available at all the servers. However, now servers N1 and N2 are overloaded, while server N3 is underloaded. The job is launched at server N1. The migration time between servers is varied from 100ms to 300,000ms. As is expected, we see that for low migration times, most of the tasks migrate to the underloaded server N3 due to adaptation. The number of such migrations due to adaptations decreases as the migration time increases, as the higher cost of moving the task to a remote server makes it less advantageous to migrate. The completion times of the task graphs are in between those in Table 1 and 2.

Table 4 (shown at the end of the paper) shows results for a different scenario. The servers N1, N2, and N3 now have different resources. N1 has 3 copies of r1, 3 copies of r2, and 3 copies of r3; N2 has 1 copy of r1, 2 copies of r2, and 3 copies of r3; and N3 has 2 copies of r1, 2 copies of r2, and 3 copies of r3. Servers N1 and N2 are overloaded and server N3 is underloaded. The job is initially launched at N1. The migration time is again varied from 100ms to 300,000 ms. Now a task may migrate because the current server does not have all the resources that it needs, or it may migrate due to adaptation. The first type of migration is unavoidable, even with very high migration time. As expected, the number of migrations due to adaptation decreases as the migration time increases. The completion time of the tasks compare favorably with those in Table 2.

An analysis of the system logs shows some interesting points. First, for low migration times, it is often found that an agent at the beginning migrates from a server X to another server Y and back. This is a useless migration. This happens because initially the agent has no knowledge of the average waiting time at the remote server Y, and optimistically assumes it to be 0. However, on arrival at Y, if Y is overloaded, it may find that the expected waiting time at Y is much larger, and it was better to move back to X. This extra migrations for a number of tasks add an unnecessary component to the completion time of the job. We are planning to implement a system wide dynamic resource monitoring system that can provide up-to-date information about the load condition of the remote servers. This is expected to significantly improve the performance.

The second point we noted from our analysis is that in most cases, when the simulator reported an expected waiting time of *w* to an agent, the agent did acquire the required resources within that time. However, there are quite a few cases where the agent waited much less or much more to acquire the resources. This is because of the simple scheduling policy we used for the resources. An agent requiring a set of resources simply tries to acquire all the resources within a lock; if it acquires all the resources, it goes ahead and executes the task, else it just retries (if it decides not to migrate). Thus, there is no fair scheduling policy for allocation of the resources. An agent may get the resource within a very small number of tries even at an overloaded server, or it may have to wait for a long (longer than the expected waiting time) time even at an underloaded server. Implementing a fair, bounded-wait scheduling policy can circumvent this problem.

4 RELATED WORKS

Although adaptation has been studied in the context of intelligent agents in AI [10], there has been little work in modeling adaptive mobile agents. Goodwin [2] presents a model of a robotic agent and the environment in which it operates. The concept of a Mechanism to interact with the environment is proposed by them. Our model borrows some concepts and terminology from them. However, their model does not consider adaptation. Luck and d'Inverno [5] first uses the idea of motivation as defined by Kunda [4] to define autonomy of an agent. Their work is also restricted to specification of static agents and does not model adaptation explicitly. Decker et. al. [1] describe a financial portfolio management system based on adaptive agents that use cloning for query load distribution as the principal form of adaptation. Though their work does not explicitly use mobile agents, they mention how mobile agent frameworks can be used to send cloned agents to another processor.

5 CONCLUSION

In this paper, we have presented a model of an adaptive mobile agent. We have also presented preliminary results for a task scheduling system using adaptive mobile agents to show that integrating adaptation into mobile agents can be beneficial in distributed applications. The current implemention is simple, and can be improved in many ways. This includes dynamic resource discovery and monitoring which will allow for better adaptation to improve performance, better resource scheduling policies for guaranteeing bounded wait time of tasks, and better adaptation policies. The issue of bounded wait time is particularly important if the system is to be applied to real time tasks with deadlines. We are also looking at ways to model the learning of new adaptation methods and dynamically changing the motivation degree function based on system history.

REFERENCES

1. Decker, K., Sycara., K., and Williamson, M., "Intelligent Adaptive Information Agents," AAAI'96 Workshop on Intelligent Adaptive Agents, 1996.
2. Goodwin, R., "A Formal Specification of Agent Properties," Journal of Logic and Computation, 5 (6).
3. Gray, R., Kotz, D., Cybenko, G., and Rus, D., "Mobile agents: Motivations and state-of-the-art systems," Technical Report TR-2000-365, Department of Computer Science, Dartmouth College.
4. Kunda, Z., "The case for motivated reasoning," Psychological Bulletin, 108 (3), pp. 480-498.
5. Luck, M., and d'Inverno, M., "A Formal Framework for Agency and Autonomy," First International Conference on Multi-Agent Systems, 1995, pp. 254-260.
6. Minar, N., Kramer, K. H., Maes, P., "Cooperating Mobile Agents for Network Routing," Software Systems for Future Communication Systems, Springer Verlag, 1999.
7. Pham, V. A., and Karmouch, A., "Mobile Software Agents: An Overview," IEEE Communications, 36 (7), pp. 26-37.
8. Shehory, O., Sycara, K., Chalasani, P., and Jha, S., "Agent Cloning: An approach to Agent Mobility and Resource Allocation,", IEEE Communications, 36 (7), pp. 58-67.
9. Tripathi, A., Tanvir, A., Kakani, V., Jaman, S., "Distributed Collaboration using Network mobile Agents," Univ. of Minn.
10. Woolridge, M. J., and Jennings, N. R., "Intelligent Agents: Theory and Practice," Knowledge Engineering Review, 10 (2), 1995.

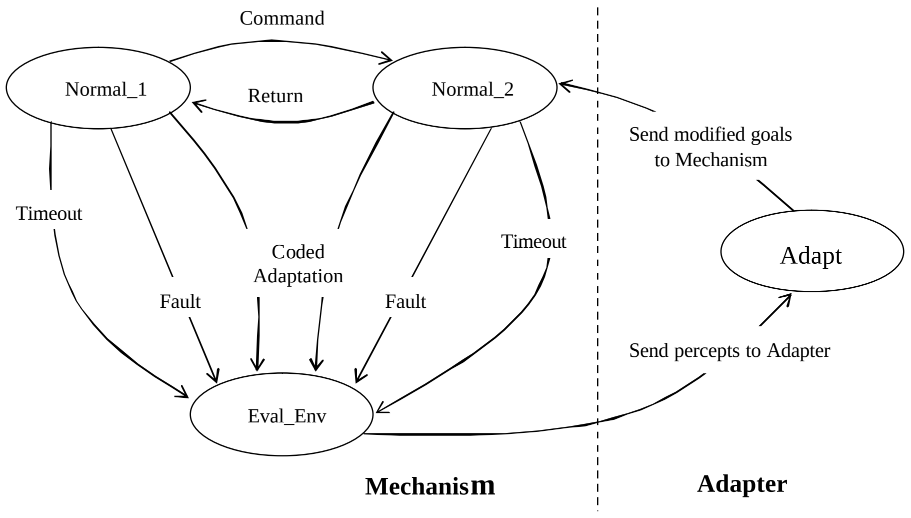

Figure 2: State diagram of an adaptive agent

Task Graph	No. of Tasks	Migration Time (ms)	No. of Migrations	Tasks completed at N3	Completion Time (sec.)
T1	35	100	36	23	110
		500	22	31	80
		1000	15	26	82
		2500	5	33	99
		300,000	0	0	326
T2	39	100	35	33	83
		500	34	33	101
		1000	20	31	102
		2500	13	35	143
		300,000	0	0	477
T3	40	100	22	30	79
		500	27	29	99
		1000	19	35	116
		2500	13	30	130
		300,000	0	0	356

Table 3. N1,N2 overloaded, N3 underloaded. All servers have same resources

Task Graph	No. of Tasks	Migration Time (ms)	No. of Migrations	Migrations due to Adaptation	Completion Time (ms)
T1	35	100	44	41	77
		500	38	35	90
		1000	39	36	107
		2500	11	7	104
		300,000	9	0	866
T2	39	100	26	13	145
		500	33	21	183
		1000	31	18	231
		2500	23	12	204
		300,000	8	0	742
T3	40	100	44	42	89
		500	38	37	93
		1000	37	35	99
		2500	16	14	140
		300,000	5	0	746

Table 4. N1,N2 overloaded, N3 underloaded. Servers have different resources