Toward a Methodology for the Development of Network Management Solutions: a Study on Mobility, Autonomy and Distribution in Agents

Hendrik Teixeira Macedo, Carlos A. G. Ferraz and Geber Lisboa Ramalho

Centro de Informática, Universidade Federal de Pernambuco 50732-970, Recife, PE, Brazil

htm@cin.ufpe.br

(Recebido em 20 de maio de 2005; aceito em 16 de junho de 2005)

Current computer networks require an intricate management activity in order to provide high quality of service to system users. Usual automation tools follow a client-server approach, which centralizes the processing. Such management kind lacks flexibility and fault tolerance and generates high network traffic. A promising alternative approach is based on intelligent agents capable of taking decisions more autonomously and migrating between devices, allowing a more distributed management. This work discusses the impact that properties such as mobility, autonomy and distribution in agents have in the development of management solutions for corporative networks. Considering some metrics (processing time, CPU consumption, space savings), we have implemented and made an evaluation of different multi-agent architectures varying the levels of these properties. As case study we used the disk space management problem in UNIX/NFS machines. A simulator was developed to carry out experiments with higher control. The results can be used as a guide toward a methodology for construction of management solutions based on mobile, autonomous and distributed agents.

Keywords: network management, intelligent agents, mobility

1. INTRODUCTION

Agent-based systems have recently gained significant attention in several computer science fields such as software engineering, human-computer interfaces, and network management, and the provision of intelligence implies dealing in an adaptive way with unexpected changes in the environment [1].

Mobile agents [2] introduce a new software and communication architecture, consisting of executable codes that "travel" between networked machines in order to process data locally. Unlike the Client-Server (CS) approach [3], there is no need to bring intermediate data across the network, and thus a significant amount of network bandwidth usage and communication delay can be avoided. This idea has been popularized in recent years in the area of network management [4], [5].

The automation of network management activity implies delegating actions that would be manually accomplished by human managers to computational proceedings. Automation is required since management activity is usually very repetitive and incessant which may cause imprecision. In addition, decisions demand time and sub-utilization of human resources. Interruption in the availability of computational resources or even any inconsistency in the state of network transactions can cause severe injuries to the organization systems.

Current tools for network management are built according to client-server (CS) approaches and are strongly characterized by a centralized management. Agents located in devices to be managed verify the state of devices periodically and send messages to the network management station (NMS) (machine used by the human manager) which is responsible for monitoring the devices, identifying problems and turning on proceedings to fix any sudden operation change. Such centralization overloads the NMS, increases the network traffic and limits the management flexibility.

Using more sophisticated agents in network management tools could minimize the effects of centralization. An agent can, for instance, move its own code and execution state from one machine to another and take decisions autonomously [7]. Moreover, properties like mobility and autonomy would help to distribute the management, what is desirable. Unfortunately, solutions that try to implement such type of agent still suffer from a big problem: the lack of a precise methodology that indicate when and how to endow agents with mobility and autonomy. Precise answers to questions like *Mobile or static agents? What is the adequate autonomy degree? How many agents should be used? How different should be the agents?* provide the initial subsidies for setting up an agent-driven methodology for development of network management solutions. Indeed, these answers could help to define parameters to assess the potential advantages and disadvantages of each agent's implementation aspect for a given management problem.

With the study of some agents' architectures that differ on the utilization levels of the provided properties, we think it is possible to furnish some items that would aid the design of network management systems. Through a reflection on three aspects namely, *mobility*, *autonomy* and *distribution*, we have defined types of agents and organizations of agents by the combination of those types. Afterward, we have adapted these types and organizations to a particular case of network management domain, the disk space management in UNIX/NFS networks. Finally, we have implemented a generic simulator for carrying out the experiments whose results could be used as guide for the construction of a well-based methodology for the design of network management systems.

The remainder of this paper is structured as follows. Section 2 exposes traditional approaches for network management and agent-based ones. In section 3 we discuss some open issues regarding the implementation of those aspects in agent-based systems. Section 4 shows the steps we have follow since the definition of some types of agents till the implementation of the simulator, that is, describes our work methodology. The experiments and its results are shown in section 5. In Section 6 we draw some conclusions and point out directions for future works.

2. NETWORK MANAGEMENT APPROACHES AND AGENTS

Usually, most of the network management knowledge resides in the human manager. He knows the management politics, the demands of the systems users and its quality requirements and it is him that possesses the operational knowledge to transform these demands in actions. The operational management is made through the monitoring of the status of the system, analysis of the current situation and triggering of appropriate commands for errors correction or optimization of the resources usage. Such transformation of management politics in correction actions usually happens in a reactive manner and exactly in the instant that a mistake requires the corresponding correction action [7].

Traditional Approach

The most used approach for network management is proposed by IETF (Internet Engineering Task Forces) and it is based on the Simple Network Management Protocol (SNMP) [9]. The approach follows the widely used client-server model where a centralized managing entity (NMS – Network Management Station, operated by the human manager) interacts with SNMP agents in execution in the devices of the network. Each SNMP agent stores the device's information in a local information base called management information base (MIB) [10]. The NMS acts as client of such agents that control the remote access to its local MIB, requesting information of the status of the network devices through some SNMP protocol primitives for exchange of messages. The structure of SNMP agents is very simple and they do not execute management actions in its local data. The maximum initiative they take is the dispatch of messages to the NMS when a specific event happens (the sudden change in the status of a component from "active" to "inactive", for instance). The NMS is entrusted of executing the resolved management action by the human manager. Such typical client-server interaction leads to generation of high network traffic and NMS overload, where all the computation is accomplished in fact. Furthermore, solutions based on those approaches have poor flexibility and scalability [8].

Agent-based Approach

Traditional approaches as the mentioned so far use a very simplistic concept of agents that function basically as a sensor. Possibly, endowing agents with characteristics such as mobility, reasoning, coordination ability, etc., they could give a larger contribution for a distributed, flexible and less painful management for the human manager.

A mobile agent (MA - mobile agent) is a computational process which can bring its own execution code, the data it processes and even the execution status from a device to another. Client-server models assumes that the important thing is to migrate the data to where the program which will process them is located, while those based on mobile agents consider that migrating the program to the place where the data are located is the best approach. As a result, interactions between clients and servers are not any more done remotely; they are done locally, reducing the network traffic, increasing the systems' robustness, and providing a larger flexibility, once a certain service is not statically tied up to a specific machine.

A reasoning agent has been classically implemented through deductive inference (or inductive [11]) mechanisms that allow the agent to choose the most appropriate actions starting from its perceptions, its goals and its knowledge [7]. Such ability of reasoning logically is a important characteristic in order to provide a larger autonomy degree to the agent. Indeed, the autonomy can improve the benefits of agents' mobility in the network management [5]. For example, a mobile agent can make dynamic decisions such as finding the next destination, optimizing the travel plan, and detecting link failures, as the agent travels around the network.

In order to follow the decentralization tendency that is highly recommended nowadays to the automation of network management systems [8], it is necessary thinking not just in a lonely agent, but in a group of them, called multiagents systems [10]. When the problem requires the action of physically distributed entities, that is the case of the network management, ones should opt for a multiagent solution. In order to agents be able to co-act, good mechanisms of group coordination are made necessary.

Those three characteristics described above have been studied, in the academic domain, to be used in the modeling, development and/or usage of more sophisticated agents in the network management field [13], [14], [15]. However, despite of the great potential of the agents, the current researches seem not being collaborating much. Little profit has been taken from the couple mobility-autonomy, for example. Besides, the solutions are still based on the centralized control and there is not yet an accurate methodology for the design of agent-based network management solutions.

It seems to be clear that there is not a guide for a coherent usage of characteristics such as mobility and autonomy in the development of network management solutions. The way in which researches are being driven, does not really contribute to the development of a technique because they are not coordinated, they don't investigate the circumstances of the usage of the agents' characteristics and they contemplate only some specific network management cases [16].

3. OPEN ISSUES

We thought the discussions on the development of less centralized approaches, where autonomous entities are responsible for the management of such a resource, should be moved toward a new context, where the network should be faced as one of those resources to be explored and managed. That new vision implies a change of human manager attitude and the increase of the complexity of the automation tool and also upsets standardized management model, where a central NMS is of speaking specific network protocols and interacts with the managed resources. Why not having a distributed management system where there is not a central station and where the intelligence simply emanates from a set of intelligent agents that collaborate to solve a specific problem? If on one side, the distribution of the management of the resources decreases the need for human intervention, for other, it increases the need for organization of intelligent agents. Hence, it is important to make an evaluation of the inclusion of agents' properties like mobility and autonomy, for instance, in such a way they can be used in a consistent manner.

In a previous work [4] we have implemented three different architectures that combine these characteristics and we have also made some performance measurements to evaluate the impact of these parameters, primarily, regarding the network traffic. We have noticed that although such architectures have been potential solutions for network management, the results has shown that much more different architectures are necessary to provide convincing results. The importance of a design methodology becomes clear. There are some open issues around the use of Artificial Intelligence (AI) that need to be answered as a first step toward such methodology: *Should it be used static or mobile agents? Which the ideal degree of mobility? How complex should the agent's code be? Which is the ideal degree of autonomy? Which the ideal number of agents in the solution?*

Although the well-known potential benefits from the use of mobile agents such as space savings, decrease of network traffic, robustness and fault tolerance and others [5], its employment is circumstantial and depends on the kind and the level of the network management activity. The big challenger is to discover which these circumstances are. When there is not a strongly need of frequent management activities, maybe the use of mobility do not compensate the cost of supplying the required mobility middleware. When there is a great variation on the status of managed devices, where simple problems occur regularly, the use of mobility may be more suitable. However, how to deal with an unstable network? What should be done when some devices need to be managed more often?

By the way, autonomy can augment the benefits of mobility by reducing the need of human expertise during installation and operation of the management solution. On the other hand, great autonomy degree requires complex agent implementation, with more coded rules. So, would it be interesting to create agents with different specialization and functionality? In situations where activities are quite complex and can be split into sub-activities, the use of specialized agents could be interesting. To be precise, activities as *perception*, *decision* and *execution* can be imputed to individual agents, combined for the whole proposes.

In a management domain where variations on the status of devices are continuous, it may be interesting the use of a larger number of agents. The agents can be spread in the network and become in charge of a sub-network. On the other hand, a greater number of agents leads to a larger need for coordination activity. The number of agents used a solution can also be directly related with its autonomy. Agents with restricted power, and more specialized, need to cooperate with other agents in order to solve problems: and that is another coordination issue.

4. METHODOLOGY

The exposed issues above turn evident the hardness of projecting and developing agent systems for truly distributed network management solutions. In order to contribute to the area, we have followed a work methodology which consists of four steps, basically: definition of some types of agents and organizations of agents, establishment of comparisons criteria, a case study for adapting the types of agents and organizations, and the construction of a simulator to carry out the experiments.

4.1. Types of Agents

The previous discussion has shown that there are at least two properties to be taken into account when developing agent-based network management tools:

- *Mobility*: an agent may be mobile, static local or static connected. Static local means that the agent is fixed in a device and its actions are strictly local. Static connected means that the agent is fixed in a device but may be connected to other devices through remote calls;

- *Autonomy*: an agent may holds the whole management knowledge or be specialized (perception, decision or execution, for example).

These properties may be represented by two sets ("Mobility" and "Autonomy") and the resulting elements of the Cartesian product are the types of agents we have defined and named (Fig. 1).

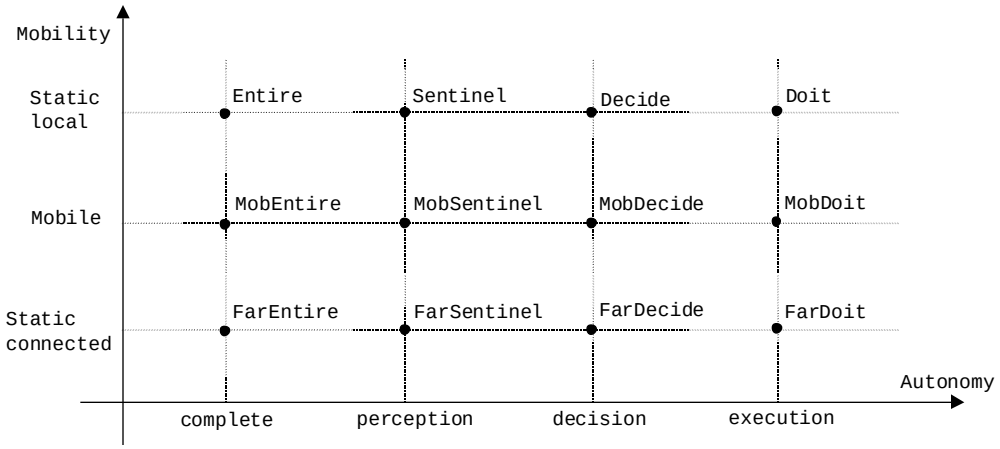

Fig. 1. Types of agents from properties

The Entire type defines static agents that hold the whole management knowledge and act only on the device it is located. Sentinel, Decide and Doit types define specialized static agents in charge of monitoring the status of managed device, defining the most convenient actions to solve the problem, and executing those actions, respectively.

Mob types define agents with similar functionality of each type described before, but with the ability of moving between devices. Their actions are also locally.

The Far types also define static with similar functionality of each type described before, but although they are static, they do not act just locally. They can work on others devices through remote calls.

Besides these 12 types of agents, we may also have *associated types*. Depend on the management problem domain, it could be more suitable having a static agent that amasses both functionality of deciding and executing, for example. That agent would belongs to the associated type DecideDoit.

Organizations of Agents

A multi-agent architecture can be obtained by combining several of those types of agents. Because the number of combinations may be very large, we have introduced the concept of *organizations of agents* to gather kindred types of agents. An organization of agents works as an intermediary level and helper to the definition of a multiagent architecture.

To be concerned about the number of possibilities there are, suppose the example of a network with just three manageable devices. The Fig. 2 illustrates 6 possible architectures.

From left to right and from top to bottom, the first is composed by two agents of the type MobEntire (ME), the second is composed by an agent of the type MobSentinel (MS) and one of the associated type MobDecideMobDoit (MDMD), the third is composed by an agent of the type FarSentinel (FS) and one of the associated type FarDecideFarDoit (FDFD), respectively, the fourth composed by three agents of the type Entire (E), the fifth composed by an agent of the type MobSentinel (MS) and two of the associated type MobDecideMobDoit (MDMD), and the last one composed by three agents of the type Sentinel (S) and one of the associated type MobDecideMobDoit (MDMD), respectively.

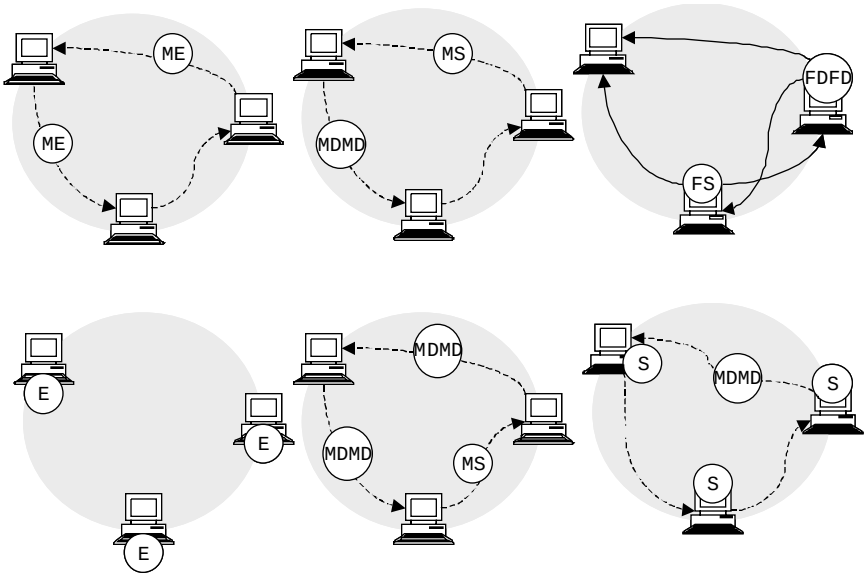

Fig. 2. Different multiagent architecture compositions

This example shows the hard work of deciding the more appropriated types of agents for a specific network management solution and also the number of agents to be used. Since it is not possible to treat all the possibilities, we have made an evaluation of each type and identified the most rational compositions (Table 1).

Table 1. Organizations of Agents

Organizations of agents	Types of Agents
LocalMonoComplete	Entire
MobMonoComplete	MobEntire
ConectMonoComplete	ConectEntire
LocalThreeSpec	Sentinel, Decide, Doit
MobThreeSpec	MobSentinel, MobDecide, MobDoit
ConectThreeSpec	FarSentinel, FarDecide, FarDoit
MobConectMixed1	FarSentinel, MobDecideMobDoit
MobConectMixed2	MobSentinel, FarDecideFarDoit
MobLocalMixed	Sentinel, MobDecideMobDoit
MobTwoType	MobSentinel, MobDecideMobDoit

4.2. Case Study: Disk Space Management in UNIX/NFS Networks

We have chosen a particular problem of the Accounting Management OSI Functional Area [15] to employ the ideas exposed prior. The disk space management is a very important activity for the good operation of a corporative network since several applications and network services, such as logging and e-mail services, have high dependence on the free space in disk.

The Network File System (NFS) is a transparent environment for sharing and distributing files in UNIX networks. It implements a client-server environment, which extends the common access functionalities to files in UNIX so that machines can share portions of their local filesystem. This contributes to the distribution of storage resources and to remote files sharing.

Each storage device (local or imported from other machine in the network) used to compose the virtual directories tree of the machine is called *partition*.

A common problem in such kind of environment is the over utilization of an individual partition. Specifically, the number of files in a partition can grow until take the whole available space in disk. The seriousness of such event depends on the type of the partition. The lack of control in a partition of an e-mail area (“/var/mail"), for instance, can lead to the unavailability of such service, making electronic mail messages be rejected or discarded.

4.2.1. Activity Modeling

The management automation for this activity consists in (1) provideg periodicals checks on the utilization percentage of a partition, (2) determine if it is super-used (it has surpassed a threshold previously specified by the network manager), (3) determine appropriate correction actions, and (4) apply them. Usually, these actions consist of several different operations on the existing files. One may seek for very big or useless files and remove them; files may be compacted to save space, may be moved to another partition even it is a remote one. The choice should be taken according to some criteria as files lifetime, duplication of files, files types, to point up some. For this reason, a management tool for the disk space problem should include the following operations:

1- *Partition Classification (PC)*: This operation identifies all the partitions in a machine's hard disk and classifies them according to some predefined partitions types (ex. "/", "/usr", "/var", etc.).

Pseudo-code:

```
getFilesystems() {
     List existing partitions in machine with "/bin/df -kl"
    Keep the list in memory
    For ∀ lines of the list do:
        Identify the size,
                 the partition's utilization,
                 the partition's name
        If name = "/": partition classified as ROOT type
        If name = "/usr": partition classified as USR type
        ...
}
```

2- *Utilization Checking (UC)*: For each partition's type there is a correspondent utilization threshold. We have provided a text file with these limits.

Pseudo-code:

```
checkUtil() {
    For ∀ classified partition ρ do:
        Read the thresholds' file φ
        Seek φ for the threshold for ρ
        If the utilization % of ρ ≥ threshold:
            ρ is super-used
}
```

3- *File Classification (FC)*: This operation consists of a depth-limited search [7] on the partition's sub-directories tree. As the files (their name consist of the whole path) are being identified they are classified according to the directory it is located and/or information obtained with the UNIX system call "/bin/file <file_name>".

Pseudo-code:

```
classifyFiles() {
    For ∀ super-used partition ρ do:
        ν ← accumulate(ρ)
        While ν is not empty:
            l ← first line of the last position of ν
            If l is last line of the list
                Remove last position of ν
                If ν is empty
                   end-of-while
            If l begins by the letter 'd':
                It is a directory
                accumulate(l)
            else if l begins by the character '-':
```

```
                It is a file
                Get file information with "/bin/file"
                Classify file
}
accumulate(path p) {
    List the files|directories of p with "/bin/ls -lAc"
    Store the list in a vector in memory
}
```

4- *Action Decision (AD):* This operation defines, according to some rules, the actions that should be taken for each classified file in order to solve the over-utilization problem. For example, it could define that a temporary file, which has not been accessed for a week, must be removed, or a big image file must be compressed. Those production rules have been described in a similar format to the First Order Logic (FOL) [7]. All the rules have been codified using the JEOPS inference engine's format [17].

Examples of some Production Rules:

$\forall p,f$ Partition(p) Λ File(f) Λ (Type(f) = CORE) $\rightarrow$ Remove(f)
$\forall p,f$ Partition(p) Λ File(f) Λ (LifeTime(f) > 5) Λ
(Type(p) = TMP) $\rightarrow$ Remove(f)
$\forall p,f$ Partition(p) Λ File(f) Λ (Size(f) > 5000000) Λ
(Type(f) = MAIL) Λ (Type(p) = EXPORT)$\rightarrow$ Compact(f)

Pseudo-code:

```
reasoning() {
    Insert the super-used partition in Knowledge Base (KB)
    Insert ∀ classified files of that partition in KB
    Run KB
}
```

5- *Action Execution (AE)*: This operation is responsible for executing the inferred actions. This is done with the aid of UNIX system calls.

Pseudo-code:

```
executeActions() {
    Removing: "rm <file_name>"
    Compacting: "gzip <file_name>"
    Compacting and Recreate: "gzip <file_name>" and
                             Recreate it empty
    Moving: export partition with available space in purposed
            machine, mount remote partition in the former
            machine and move the file to the remote partition
}
```

4.2.2. Adapting Types of Agents

In section 4.1 we have defined 12 types of agents. They were adapted to the case study. Depending on the functionality, the agents are in charge of executing some of the operations described so far. Fig. 3 shows the architecture of the MobEntire type.

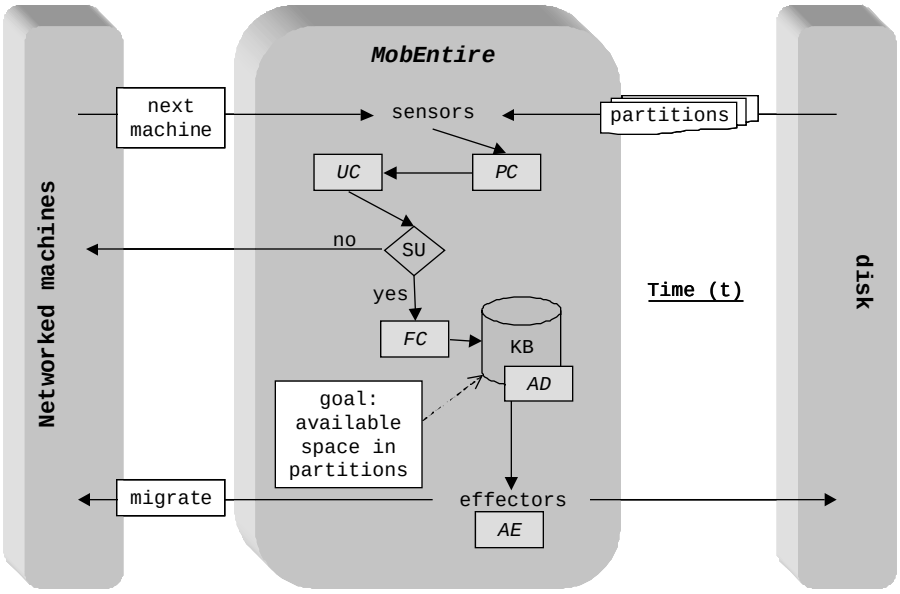

Fig. 3. The architecture of a MobEntire agent

4.3. The AgsAge Simulator

In order to better accomplish a larger number of experiments and in a more controlled way we have developed a management simulator. The great hardness to obtain the exclusiveness of a network for actual experiments and the performance variations that a network usually has could commit the results. The simulator allows variation on the number of network devices and composition of different network sceneries. This is desirable since we need to test the performance of different multi-agent architectures and the related cost of each operation type in the network: remote messages and local ones, code migration, multicast of messages, CPU consumption, processing time, disk space consumption, etc.

4.3.1. Organization and Operation.

AgsAge [18] works independently of the network management application. In other words, it has a modular architecture which enables the specification of a whole simulation environment for any management activity. Such modularity is achieved by means of an API (Application Program Interface) that allows the designer to independently specify organizations of agents, types of agents, type of devices and the management domain.

In our case, we have used the to implement the problem of disk space management. The simulator operation for the case study consists of three main steps:

1- Addition of machines. One must specify the type of each added machine. We have provided five possible types according to the level of partitions usage;

2- Selection of the organizations of agents. One may use one of the five organizations of agents provided (LocalMonoComplete, MobMonoComplete, ConectMonoComplete, MobLocalMixed, MobTwoType);

3- The distribution of the agents. Agents (belonging to the selected organization) should be distributed throughout the network machines.

When the simulation begins, it will be fired several threads that will execute, one for each existing agent. In the case of the MobLocalMixed organization, for instance, Sentinel agents remain monitoring the usage status of the partitions of machines they are placed. When they

detect over-utilization problem in any partition, it requires a MobDecideMobDoit agent action, doing a message multicast to all other agents of this type that are distributed in the network. Fig. 4 illustrates a simulation process.

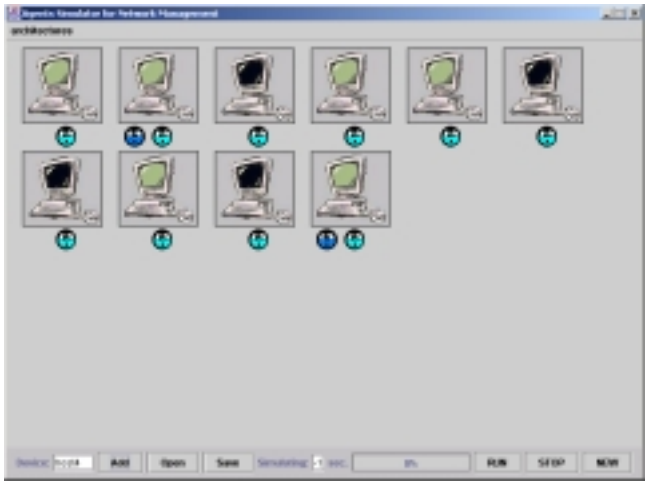

Fig. 4. Simulation with several Sentinels and two MobDecideMobDoit agents

4.3.2. Calibration

To calibrate the simulator we must measure the impact (in terms of processing time, CPU consumption, etc.) that each of the operations (section 4.2) causes in a network and to embed the simulator code with the gotten values. This way it is possible to evaluate the performance of the simulated multi-agent architectures. Those operations have been implemented in three different ways: (1) for local execution, (2) for remote execution through UNIX *Remote Shell* (RSH) and (3) for remote execution through *remote objects*. Codes for the measurement of agents' migration cost and for the cost of agents' inter-communication have been also implemented. We have chosen the distribution's platform Borland's Visibroker (CORBA [19]) with IDL compiler for JAVA to implement the remote objects. The mobility middleware used was the ObjectSpace Voyager [20]. Each operation was executed 100 times.

5. EXPERIMENTS AND RESULTS

We have accomplished some experiments with five different multi-agent organizations (*LocalMonoComplete*, *MobMonoComplete*, *ConnectMonoComplete*, *MobLocalMixed*, *MobTwoType*), varying the number of agents and the scenario (i.e., different number of machines with different levels of disk usage. In all cases, we have fixed a limit time for the simulation. We use the term "visit" to indicate that the sub-tasks *partition classification*, *utilization checking* and *files classification* of a machine have been done, either by mobile or static agents.

We have firstly simulated a medium size network (30 machines) without over-utilization problems to verify how well the machines have been visited. During the given experiment time span, the organization *ConnectMonoComplete* (either in RSH or Remote Object implementation) have visited only 50% of machines. This undesirable behavior is due to the fact that *FarEntire* (agents that solves the whole problem remotely) is probably an inappropriate model. On the other hand, the organizations *LocalMonoComplete* and *MobLocalMixed* have visited all machines more times than necessary, since the Sentinel agents perform "visits" continuously (Fig. 5). The other two organizations have done a reasonable number of visits. Fig. 5 also shows that the *MobTwoSpec* organization with only one *MobSentinel* and the *MobMonoComplete* organization with only one *MobEntire* have had an equivalent behavior. But three *MobSentinel* agents have made more visits than three *MobEntire* agents. This can be

explained by the fact that the migration time of a *MobSentinel* agent is smaller than the *MobEntire* one, since the former has a smaller code.

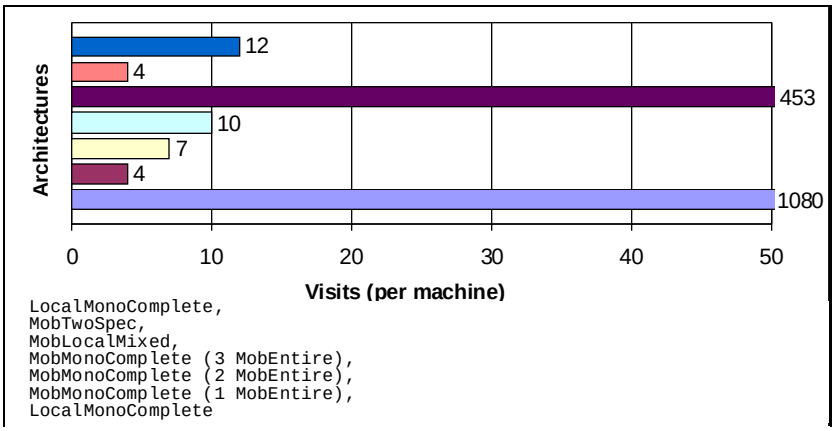

Fig. 5 - Number of visits in a 30-machine network without of super-utilization problems

In networks with low rates of over-utilization problems, organizations that implement static agents have obviously high visit rate per machine, consuming too much cpu. On the other hand, organizations containing mobile agents face the problem of high number of unnecessary migrations in network. Fig. 6 shows the results of an experiment in a 15-machine network, where 4 machines had super utilization problems. It is important to notice that increasing the number of agents *MobEntire* in the organization *MobMonoComplete* the processing time and management latency decrease, as it was expected. However, in this case, the number of migrations in the network increases in a higher proportion. This means that there was a great growth in the number of unnecessary visits (machines without problem).

architectures	time	latency	suc(%)	migKB	mig	msg
LocalMonoComplete	34	0,38	100,00			
ConectMonoCompleteOR	300		25,00			
ConectMonoCompleteRSH	300		25,00			
MobMonoComplete (1 MobEntire)	132	52,35	100,00	14		
MobMonoComplete (2 MobEntire)	75	23,55	100,00	38		
MobLocalMixed (1 MobDecideMobDoit)	89	47,05	100,00	4		915
MobLocalMixed (2 MobDecideMobDoit)	72	22,08	100,00	4		430
MobTwoSpec (1 MobDecideMobDoit, 1 MobSentinel)	89	47,70	100,00	4	40	18
MobTwoSpec (2 MobDecideMobDoit, 2 MobSentinel)	56	32,10	100,00	4	64	72
MobTwoSpec (2 MobDecideMobDoit, 4 MobSentinel)	58	32,10	100,00	4	207	732
MobTwoSpec (3 MobDecideMobDoit, 2 MobSentinel)	57	29,98	100,00	4	76	6
MobTwoSpec (4 MobDecideMobDoit, 2 MobSentinel)	60	30,98	100,00	4	87	8

Fig. 6 - Results for a 15-machine network where 4 machines had super-utilization problems. Time = necessary processing time for finishing management, latency = management latency average, suc = percentage of machines that has been visited, migKB = migrations number of KB-agents, mig = migrations number of agents, msg = number of messages between agents

In organizations such as *MobLocalMixed*, the action of Sentinels agents located in each machine optimizes the management since they are responsible for the sub-task files classification (the most time costly operation) while *MobDecideMobDoit* agents are working on actually solving problems. In the organizations with *MobEntire* agents as the *MobMonoComplete* one, such agent has to process all the operations; there is no previous processing. However, the communication overhead is very larger in that then it is in this one: as soon as the Sentinels accomplish the sub-task files classification, they begin to make multicast messages for *MobDecideMobDoit* agents until they are heard. Increasing the number of agents *MobDecideMobDoit* in the organization it is clear the reduction in the number of messages, due to the larger availability of such agents.

The last four combinations of the *MobTwoSpec* organization have presented performance gains in terms of processing time and management latency, related to the first composition, but they had had similar performance results between themselves. A bigger number of *MobSentinel* agents in relation to the number of *MobDecideMobDoit* agents in the organization causes a great growth in the number of messages, since there is not enough agents of such type to attend the calls in time. We may notice also that if the number of *MobSentinel* agents is very big in relation to the number of machines with super utilization problems, there will be a high number of unnecessary migrations.

We have noticed the influence of the number of agents in the performance of an organization when we increases the number of *MobEntire* agents in a *MobMonoComplete* organization, for example. For the specific case of networks with a great number of machines with super-used partitions, 4 *MobEntire* agents have solved the problem of all machines in 65,6% of the limiting time while 2 of those agents have solved none. It was observed, however, that there had had much more migrations in the first case than in the last one (Fig. 7).

As example of the influence of the mobility, the autonomy degree and the properties correlation, the results obtained with the *MobTwoSpec* organizations have shown that an increase in the number of *MobSentinel* agents causes a great increase in the number of migrations and in the number of multicast of messages unless there is an increase in the number of *MobDecideMobDoit* agents, likewise. A balanced number of *MobSentinel* and *MobDecideMobDoit* agents have seemed to be quite functional. We have also noticed that an increase in the number of *MobDecideMobDoit* agents optimizes the processing time of the activity despite of the large communication's overhead, while a larger number of *MobSentinel* agents decreases the average management latency.

architectures	time	latency	suc(%)	migKB	mig	msg
LocalMonoComplete	82	0,41	100,00			
ConectMonoCompleteOR	500		4,35			
ConectMonoCompleteRSH	500		4,35			
MobMonoComplete (1 MobEntire)	500	237,60	47,80	11		
MobMonoComplete (2 MobEntire)	500	229,58	82,60	28		
MobMonoComplete (4 MobEntire)	328	132,50	100,00	119		
MobLocalMixed (2 MobDecideMobDoit)	488	251,08	100,00	23		102764
MobLocalMixed (4 MobDecideMobDoit)	301	163,37	100,00	23		98501
MobLocalMixed (8 MobDecideMobDoit)	234	144,96	100,00	23		54677
MobTwoSpec (1 MobDecideMobDoit, 1 MobSentinel)	500	215,61	73,90	18	18	2792
MobTwoSpec (2 MobDecideMobDoit, 2 MobSentinel)	342	150,00	100,00	23	138	5044
MobTwoSpec (2 MobDecideMobDoit, 5 MobSentinel)	280	132,96	100,00	23	464	15362
MobTwoSpec (5 MobDecideMobDoit, 5 MobSentinel)	146	70,70	100,00	23	244	7416
MobTwoSpec (5 MobDecideMobDoit, 2 MobSentinel)	266	166,88	100,00	23	85	1720

Fig. 7 - Results for a 23-machine network. All machines have super-utilization problems

The *MobMonoComplete* organization with 1 and 2 *MobEntire* agents have solved the problem of 47.8% and 82.6% of machines in network in the limit time, respectively. Folding the number of *MobEntire* agents to 4, the total processing time for resolution of the problem of all the 23 machines was 328 seconds. It can be noticed that the number of migrations have grown sufficiently in relation to the two previous situations. This is resulted in the *saturation effect* in the management. It means that how bigger the number of agents, more quickly the total management in the network is made, but when there are not any more machines to be managed, the free agents will migrate more quickly between machines.

Comparing the performance results of three different *MobLocalMixed* organizations (with 2, 4 and 8 *MobDecideMobDoit* agents) we could see the relationship between the number of agents, the mobility and the autonomy. We have noticed that there was an improvement in the processing time and in the average management latency when we increase the number of *MobDecideMobDoit* agents. However, the improvement rate seems to tend for stabilization with the increase in the number of these agents (see graphs of Fig. 8).

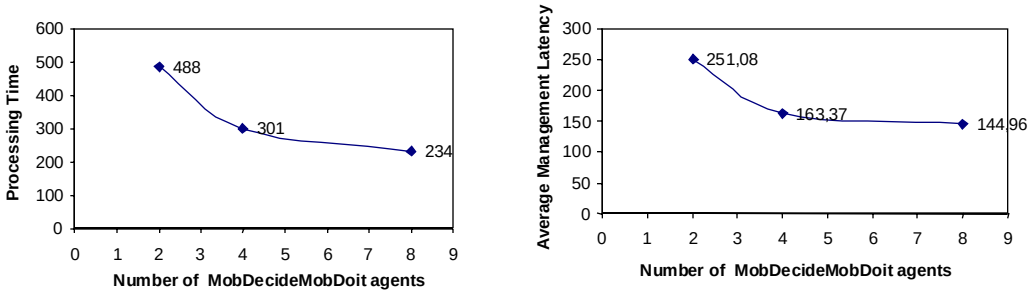

Fig. 8 - Stabilization trend for both processing time and average management latency with the increase of the number of MobDecideMobDoit agents.

The experiments results show that the usage of mobility in agents should be motivated, depending on the provided autonomy degree. Mobile agents with total autonomy had a good processing time, but had average management latency larger than an organization with specialized agents (static or mobile ones) for monitoring, decision and execution. More details in [18].

6. CONCLUSIONS

In this work, we have made a critical and original analysis of some agent's properties such as mobility, autonomy and distribution for the network management field. Unfortunately, there are little solutions nowadays that try to implement agents of such type and there is not a methodology for that.

Thus, in order to provide some guides toward the development of such methodology we have follow some steps till the accomplishment of the experiments.

From the study of some issues concerning the usage of those properties, we have defined types of agents and organizations of agents based on combinations of these types. Next, we adapted them to a particular case study: the disk space management in UNIX/NFS networks.

We have also developed a generic simulator (an arduous programming work with ~ 10200 lines of Java code plus CORBA and Voyager programming) for carrying out the experiments, which can be useful for the multiagent community's studies.

We are already working toward the adaptation of the types of agents and the organization of agents to accomplish new experiments in the simulator for another management domain.

With the results we have got, we intend to build a real multiagent-based system for disk space management based on the concept of complete distribution of the management activity.

7. AKNOWLEDGEMENTS

We are thankful to CNPq and CAPES for the financial support.

1. CHESS, D. et al. *Itinerant Agents for Mobile Computing*. IBM Research Report RC 20010, IBM Research Division, 1995.
2. HARISSON, C. G.; CHESS, D. M.; KERSHENBAUM, A. *Mobile agents: Are they a good idea?* Technical report, IBM Research Division, 1996.
3. BERSON, A. *Client-Server Architecture*. 2nd edition. McGraw-Hill, 1996.
4. ANDRADE, R. de C.; MACEDO, H. T.; RAMALHO, G. L.; FERRAZ, C. A. G.: *Distributed Mobile Autonomous Agents in Network Management*. In Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001). Las Vegas, USA, 2001.
5. BIESZCZAD, A.; PAGUREK, B.; WHITE, T. *Mobile Agents for Network Management*. In: IEEE Communications Surveys, 1998.

6. PULIAFITO, A.; TOMARCHIO, O. *Advanced Network Management Functionalities through the use of Mobile Software Agents*. In: 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99). Stockholm, Sweden, 1999.
7. RUSSEL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Pretice Hall, 1995.
8. BALDI, M.; GAI, S.; PICCO, G. P. *Exploiting Code Mobility in Decentralized and Flexible Network Management*. In: Proceedings of Mobile Agents, 1997.
9. CASE, J., FEDOR, M.; SCHOFFSTALL, M. L.; DAVIN, C. *Simple Network Management Protocol (SNMP)*. RFC 1157, 1990.
10. MCCLOGHRIE, K.; ROSE, M. T. *Management Information Base for Network Management of TCP/IP based internets, MIB-II*. In: Internet Request for Comments Series, RFC 1213, 1991.
11. MITCHELL, T. *Machine learning*. McGraw-Hill, 1997.
12. LESSER, V. R. *Cooperative Multiagent Systems: A Personal View of the State of the Art*. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, 1999.
13. BIESZCZAD, A.; PAGUREK, B. *Network Management Application-Oriented Taxonomy of Mobile Code*. In: IEEE/IFIP Network Operations and Management Symposium (NOMS'98). New Orleans, Louisiana, 1998.
14. MINAR, N.; KRAMER, K. H.; MAES, P. *Cooperating Mobile Agents for Mapping Networks*. In: Proceedings of the First Hungarian National Conference on Agent Based Computation, 1999.
15. PULIAFITO, A.; TOMARCHIO, O. *Advanced Network Management Functionalities through the use of Mobile Software Agents*. In: 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99). Stockholm, Sweden, 1999.
16. CHEIKHROUHOU, M. M.; CONTI, P.; LABETOULLE, J. *Intelligent Agents in Network Management a State-of-the-Art*. Network and Information System Journal. Volume 1, no. 1, 9-38, 1998.
17. FIGUEIRA, C.; RAMALHO, G. *Jeops – the java Embedded Object Production System*. In: M. Monard e J. Sichman (eds). Advances in Artificial Intelligence. Lecture Notes on Artificial Intelligence Series, vol. 1952, pp 52-61. London: Springer-Verlag, 2000.
18. MACEDO, H. T. *Mobility, Autonomy and Distribution in Agents for the Management of Corporate Systems*. Master's Thesis, CIn/UFPE, 2001.
19. OBJECT MANAGEMENT GROUP. *The Common Object Request Broker: Architecture and Specification (CORBA)*, Framingham, MA, 1998.
20. OBJECT SPACE VOYAGER. http://www.objectspace.com/products/voyager/