Issues in Designing Network Management Systems based on Mobile Autonomous Agents

Hendrik Teixeira Macedo[*], Geber Lisboa Ramalho, Carlos A. G. Ferraz

Federal University of Pernambuco
Center of Computer Science, PO. Box: 7851,50732-970
Recife - PE, Brazil
Phone: +55 81 271.8430, ext. 4325
{htm,glr,cagf}@cin.ufpe.br

Abstract

The use of mobile agents for network management has increased in the last years as an alternative to client/server-based approaches, which exhibit problems due to centralization. Some authors claim that agent's properties, such as autonomy (reasoning) and coordination, could be useful on the network management context. However, there are not yet any guidelines on how to put together these properties, nor an effective evaluation of the impact of this integrated approach. In this paper, we discuss some issues in endowing agents with these properties in the context of developing network management solutions. Considering some metrics (processing time, CPU consumption, disk space, etc.), we present an evaluation of different multi-agent organizations varying the levels of agents' autonomy, agent's mobility and number of agents. The experiments have been carried out by means of a simulator we have built, taking as case study the disk space management problem in UNIX/NFS networks. The results reveal some guidelines to the design of management systems based on mobile autonomous agents.

Keywords: network management, mobile agents, autonomous agents

1. INTRODUCTION

The automation of network management is necessary as some its activities are repetitive, exhaustive and require prompt reactions from humans. In fact, interruptions of computational resources and services, or inconsistencies in the state of network transactions, can cause disturbances to network systems.

The last years witnessed important advances in the network management automation, however the mainstream tools are not yet fully adequate. One of the main problems is that most management systems are built according to the client-server model, which yields to a rather centralized management. In these systems, sensors (called *Agents*), located in the devices to be managed, verify the state of the device periodically and send messages to the network management station (NMS). This station is responsible for monitoring the devices, identifying fortuitous problems, and executing actions to correct any sudden undesirable change. Such centralization may overload the NMS, increasing the network traffic and confining the management flexibility.

The use of more *sophisticated agents* in network management tools could minimize the effects of centralization [4, 15, 20]. Such sophistication could be achieved by two main

[*] Supported by CNPq (1999-2001) / CAPES (2002): BEX 0753/01-7

properties, namely *mobility* and *autonomy*. Mobile agents [1] can move their own code and execution states to other networked machines in order to process data locally, minimizing network bandwidth consumption, communication delays and failures. Autonomous agents are capable of making decisions based on their reasoning mechanisms, without the interference of humans [6]. Moreover, the combination of mobility and autonomy should help decentralize the management activity, as discussed in this paper.

Unfortunately, the works that try to implement more sophisticated agents present the following problems: little profit has been taken from the combination mobility-autonomy; the solutions are still based on the centralized control, instead of being more distributed; there is no methodology concerning when and how to endow agents of mobility and autonomy; there is no effective evaluation of the impact of using mobile autonomous agents in network management tasks.

This paper presents an original study on some important issues that arise in designing network management systems based on mobile autonomous agents. Should one use mobile or static agents? What is the adequate autonomy degree? How many agents should be used? How different should the agents be? Answering these questions would be the first step for setting up precise guidelines for agent-based solutions to network management. Indeed, these answers could help define parameters to assess the potential advantages and disadvantages of each agent's implementation aspect for a given management problem.

To find some answer to those issues, we have considered some metrics (processing time, CPU consumption, disk space, etc.) in order to evaluate different multi-agent organizations varying the levels of agents' autonomy, agents' mobility and number of agents. The experiments have been carried out by means of a simulator we have built, taking as case study the disk space management problem in UNIX/NFS networks. The results reveal some guidelines to the design of management systems based on mobile autonomous agents.

The remainder of this paper is structured as follows. Section 2 classifies approaches for network management based on agents. Section 3 discusses some open issues regarding the implementation of those aspects in agent-based management systems. Section 4 shows the steps we have followed from the definition of some agent types to the implementation of a simulator. The experiments and their results are shown in section 5. Section 6 draws some conclusions and indicates directions for future work.

2. AGENT-BASED NETWORK MANAGEMENT APPROACHES

This section discusses two current approaches in automating network management. The first one, based on a simplified notion of agents, is consolidated as the basis of several commercial tools [21,22]. The second one, based on more sophisticated agents, is rather related to academic research efforts [23,24].

2.1. Simplified agents

The most used approach for network management, proposed by IETF (Internet Engineering Task Forces), is based on the Simple Network Management Protocol (SNMP) [8]. This approach follows the widely used client-server model where a centralized managing entity (NMS – Network Management Station, operated by a human manager) interacts with SNMP agents running on the devices of the network. Each SNMP agent stores the device's information in a local information base called management information base (MIB) [9]. The NMS acts as client of such agents, requesting information about the network devices' status using some SNMP protocol primitives for exchanging messages. SNMP agents possess simple structures and they do not execute management actions in its local device. The maximum action that they take is the dispatch of messages to the NMS when a specific event happens (for example, the sudden change in the status of a component from "active" to "inactive"). It is the NMS that is responsible for executing the action chosen by the human manager. That typical client-server interaction leads to the generation of high network traffic and NMS overload. Furthermore, solutions based on such approach have poor flexibility and scalability [7].

2.2. More Sophisticated Agents

The approach mentioned before uses a simplistic concept of agents, which are basically sensors. It is believed that endowing them with properties such as mobility, autonomy, and coordination, agents could give a larger contribution for a distributed, flexible and less laborious management.

Mobile agents (MA) are computational processes capable of migrating their own execution code, the processed data and even the execution status from one device to another. The idea is to migrate the program to the place where the data are located. This approach potentially reduces the network traffic, increases system robustness, saves disk space and provides a larger flexibility, since a certain service is not statically tied up to a specific machine [4].

Autonomous agents use deductive and/or inductive inference mechanisms to choose the most appropriate actions according to their perceptions, goals and knowledge [6,10]. The ability of reasoning is an important characteristic in order to provide a larger autonomy degree to the agent. Indeed, autonomy can improve the benefits of agents' mobility in the network management [4]. 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 throughout the network. Moreover, if an agent is able to solve complex problems locally, and not just simple ones, there is no need to use the network to ask someone else to help.

Following the current decentralization tendency in network management systems [7], it is necessary to distribute the tasks among a group of agents, a multi-agent system [10]. These agents will coordinate their actions to solve problems more efficiently and more effectively.

The three characteristics described above (mobility, autonomy and distribution) 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 [12,13,14]. However, despite the potential of agents, most current research works are not fully satisfactory: little profit has been taken from the combination of mobility and autonomy; the solutions are still based on centralized control instead of being more distributed; there is no methodology concerning when and how to endow agents of mobility and autonomy; there is no effective evaluation of the impact of using mobile autonomous agents in network management tasks [15].

3. OPEN ISSUES

Considering that network management should be as distributed as possible [7], the use of more sophisticated agents, as discussed in the previous section should be encouraged.

Motivated by this new paradigm, we started a couple of years ago some implementations of mobile autonomous agents for network management tasks. Unfortunately, despite the interesting results [3], we have realized how complicated is to design and evaluate such approach. There is no a priori answer to several key questions. When to use static or mobile agents? Which is the ideal degree of mobility? How complex, in terms of reasoning capabilities, should the agent be? Which is the ideal number of agents in the solution? Should the agents communicate?

Of course, the answers to those questions cannot be given superficially. They depend on various factors such as the network configuration and size, the machines' processing power and disk space, the robustness of the network, and the management task.

For instance, concerning mobility, when management actions are not required frequently, does the use of mobility compensate the effort of installing and maintaining the mobility middleware? When the status of the managed devices varies too much, or when the detected problems are too simple, is the use of mobility still suitable? Which degree of network instability requires mobile agents? What should be done when some devices need to be managed more frequently than other ones?

Regarding autonomy, a high degree of autonomy is only achieved by complex agents, which are difficult to implement and, if they are mobile, use larger bandwidth than simple ones. In this context, should all agents have the same capabilities? Which are the specific capabilities required by the management tasks? Can these tasks be broken into simpler activities?

With respect to distribution, given that the more numerous are the agents, the more complex is their coordination, which network configuration and operational conditions do require a larger number of agents? In the case they are mobile, should these agents be restricted to a given set of networked machines?

4. WORK METHODOLOGY

The issues discussed above show how difficult it is the design and development of agent-based systems for distributed network management. In order to clarify some of these issues, we have adopted an experimental methodology which consists of the following steps: the definition of some agent types and organizations, the establishment of comparisons criteria, the choice of a case study, and the implementation of a simulator to carry out the experiments. In the rest of this section, we discuss these steps.

4.1. Defining Agent Types

The previous discussion has shown that there are two properties to be taken into account when developing an individual agent model for management tasks: *mobility* and *autonomy*.

With respect to mobility, we have decided to consider 3 possible agents: 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.

The identification of different levels of autonomy is a complicated task. We have then chosen the following types of agents regarding autonomy: an agent may hold the whole management knowledge or be specialized in perception, decision making or action execution. Fig. 1 shows 12 different types of agents obtained from the Cartesian product of mobility and autonomy possible values.

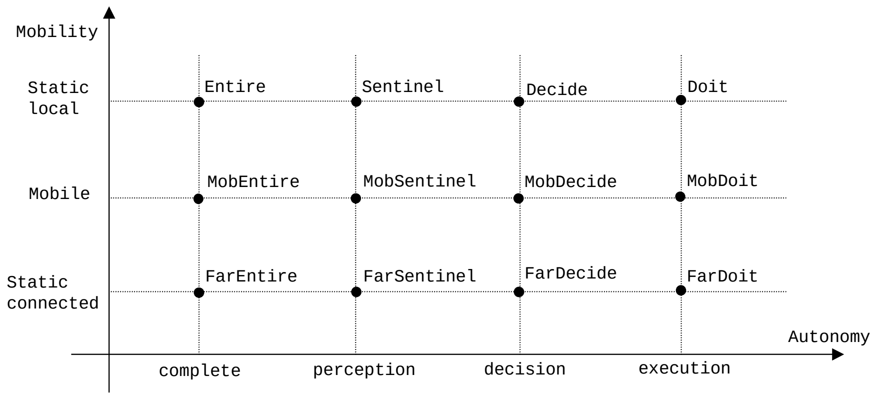


Fig. 1 - Types of individual agents considering mobility and autonomy

4.2. Defining Agent Organizations

Agent organizations can be obtained by combining different types of agents. Since the number of combinations may be very large, we have restricted the organizations to be studied for those that make sense. For instance, it is useless to consider organizations containing only perception, only decision-making or only execution agents. Furthermore, as far as the definition of organizations is concerned, we do not take into account the number of agents of the same type. An organization composed of two *MobEntire* agents has the same label as another one containing ten *MobEntire* ones. Only at the test step, this difference is considered.

Fig. 2 illustrates three possible organizations: *MobMonoComplete* (η *MobEntire*), *MobTwoSpec* (η *MobDecideMobDoit* coupled with κ *MobSentinel*), and *ConnectThreeSpec* (η *FarDecide* coupled with κ *FarSentinels* and with μ *FarDoit*). In *MobMonoComplete* organization, a *MobEntire* agent migrates between the machines, indefinitely. Since it is complete, it does not only detect problems in machines, but also chooses the actions to solve the problems and executes them. The *MobDecideMobDoit* agent, in the *MobTwoSpec* organization, it is possible to make decisions and execute the actions. *MobSentinel* detects the problems and ask for help from *MobDecideMobDoIt* accordingly. In *ConnectThreeSpec* organization, there are three Far agents, each one with a functionality. The *FarSentinel* detects the problems of the machines and asks for help from *FarDecide*, responsible for making decisions, which in turn, asks for help from *FarDoit* to execute the actions.

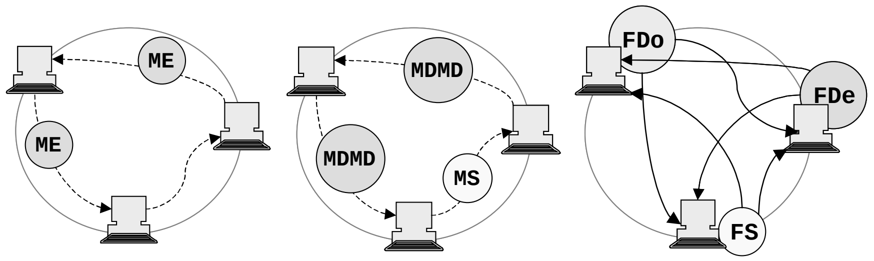


Fig. 2 - Three different multi-agent compositions: *MobMonoComplete*, *MobTwoSpec* and *ConnectThreeSpec*, respectively.

Besides the 3 organizations discussed above, we have defined 7 other possibly useful ones, as shown on Table 1.

Organizations of agents	Types of Agents
LocalMonoComplete	Entire
MobMonoComplete	MobEntire
ConnectMonoComplete	FarEntire
LocalThreeSpec	Sentinel, Decide, Doit
MobThreeSpec	MobSentinel, MobDecide, MobDoit
ConnectThreeSpec	FarSentinel, FarDecide, FarDoit
MobConnectMixed1	FarSentinel, MobDecideMobDoit
MobConnectMixed2	MobSentinel, FarDecideFarDoit
MobLocalMixed	Sentinel, MobDecideMobDoit
MobTwoSpec	MobSentinel, MobDecideMobDoit

Table 1. 10 different organizations of agents

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

We have chosen a particular problem of the Accounting Management OSI Functional Area [15] to perform the tests. The disk space management is an important activity for the good operation of a corporative network, because several applications and network services, such as logging and e-mail services, are highly dependent on storage space in disk. Furthermore, disk space management is not complex to model.

Activity Modeling

The automation of this activity consists of checking the utilization percentage of a partition, determining whether the partition has surpassed the threshold previously specified, determining the appropriate correction actions, and applying them. Usually, these actions are operations on the existing files. Files may be *removed*, *compacted* or *moved* to another, possibly remote, partition. The operation must be chosen according to some criteria such as the file size and leisure time (the time the file has not been accessed), as well as file type (i.e., temporary, e-mail, executable, image, text, etc.).

Adapting the Types of Agents

We have divided the disk management into 5 sub-tasks: *partition classification*, *utilization checking*, *file classification*, *action choice*, *action execution*. The 12 types of agents introduced in section 4.1 have been adapted to the case study. Depending on their functionality, agents are responsible for executing one or various sub-tasks. Fig. 3 shows the architectures of the *MobEntire* agent and the *MobSentinel* agent.

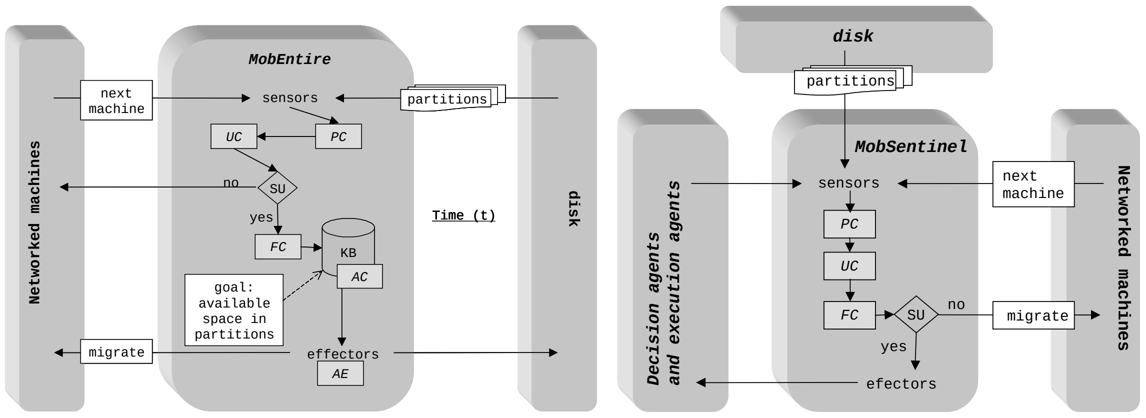


Fig. 3 - The architectures of a *MobEntire* agent and MobSentinel agent, respectively. PC = partition classification, UC = utilization checking, FC = file classification, AC = action choice, AE = action execution. Note that the files classification takes place only if the partition is super-utilized (SU).

It is important to notice that the 'action choice' sub-task was formalized using approximately 30 first order logic rules [6]. These rules have been codified in the JEOPS inference engine [16]. Fig. 4 illustrates some of such rules.

$\forall$p,f Partition(p) $\wedge$ File(f) $\wedge$(Type(f)= CORE) $\Rightarrow$Remove(f)
$\forall$p,f Partition(p) $\wedge$ File(f) $\wedge$(LifeTime(f) > 5) $\wedge$(Type(f)= TMP) $\Rightarrow$Remove(f)
$\forall$p,f Partition(p) $\wedge$ File(f) $\wedge$(Size(f)> 5000000) $\wedge$(Type(f)= MAIL) $\wedge$(Type(p)= EXPORT) $\Rightarrow$Remove(f)
$\forall$p,f Partition(p) $\wedge$ File(f) $\wedge$(Type(p)= VAR) $\wedge$(Type(f)= LOG) $\Rightarrow$Compact(f)$\wedge$ CreateEmpty(f)
$\forall$p,f Partition(p) $\wedge$ File(f) $\wedge$(Type(p)= VAR) $\wedge$(Type(f)= IMAGE) $\Rightarrow$Compact(f)

Fig. 4 - Some production rules for choosing correction actions.

The fourth rule, for example, specifies that for each partition and its respective files, if it is a /VAR partition and the file is a LOG file, then compress (compact) the file and create an empty one with the same former name. In the same manner, the fifth rule specifies that if it is a /VAR partition and the file is an IMAGE file, then compress (compact) the file. Fig. 5 below shows the JEOPS implementation for these two rules.

```
public ruleBase AgentBase {
  ...
  rule log_var {
    declarations
      Filesystem f;
      Archive a;
    preconditions
      f.isSuperUtil();
      a.getType() == Type.LOG;
      f.getType() == Type.VAR;
    actions
      a.setSituation(
       Situation.COMPACT_CREATE
                    );
      retract(a);
      modified(f);
   }
```

```
rule image_var {
  declarations
    Filesystem f;
    Archive a;
  preconditions
    f.isSuperUtil();
    a.getType() == Type.IMAGE;
    f.getType() == Type.VAR;
  actions
    a.setSituation(Situation.COMPACT);
    retract(a);
    modified(f);
}
...
}
```

Fig. 5 – Two rules of the agent's KB in the JEOPS format

4.4. 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. In fact, the difficulty in obtaining full access and exclusiveness of a network, the instability of the network and the lack of control of problems, services and demands on the network, could compromise the results. The simulator allows the composition of different network scenarios as well as the measurement of the performance of each multi-agent organization in terms of the cost of operations such as remote messages, local messages, code migration, messages multicast, CPU consumption, processing time, disk space consumption, etc.

Organization and Operation.

AgsAge [17] is an object-oriented system that works independently of the network management application. Its classes work as an API for the creation of a whole simulation environment for any management activity. The API allows the designer to define organizations of agents, types of agents, and the types of devices. The API super classes `SArchitecture, SDevice, SAgent` define attributes and methods for the definition of organizations of agents, devices, and agents, related to the desired management activity. Fig. 6 illustrates the classes' diagram of the disk space management activity. For the API complete modeling see [17].

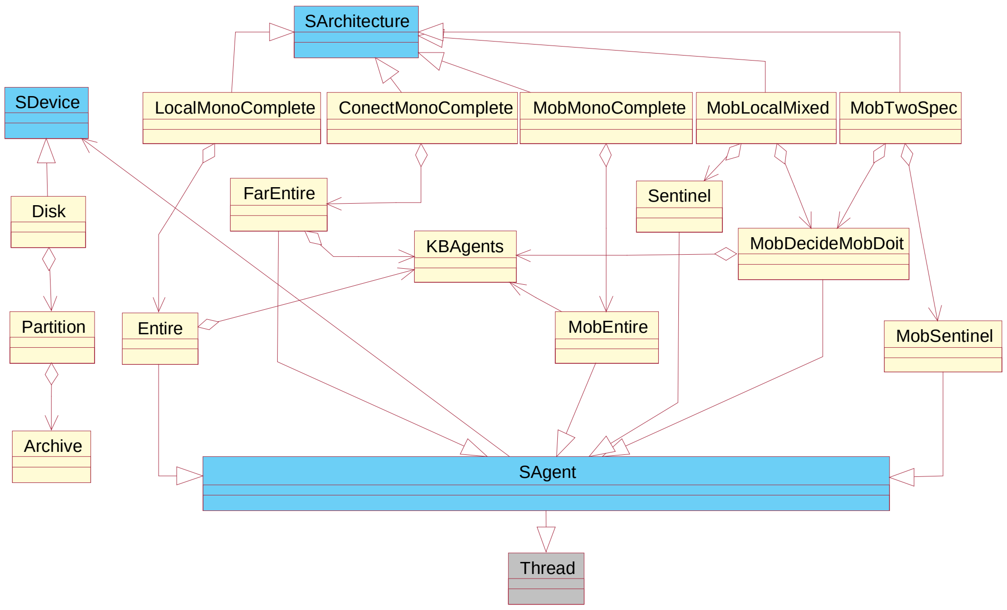


Fig. 6 – Classes' diagram of the simulator.

The simulator operation for the disk space management activity consists of three main steps:

1. *Addition of machines*. One must specify the type of each machine according to the usage level of it;
2. *Selection of the agents organizations*. There are five different organizations of agents as illustrated in figure 6. One may choose one of them for each turn;

3. *Distribution of the agents*. Agents, belonging to the selected organization, should be distributed in the machines.

Fig. 7 shows two simulation environments. The one on the left is composed of several *Sentinels* and two *MobDecideMobDoit* agents, and the one on the right is composed of one *MobEntire* agent. The machine images with a black screen represent machines with super utilization problems.

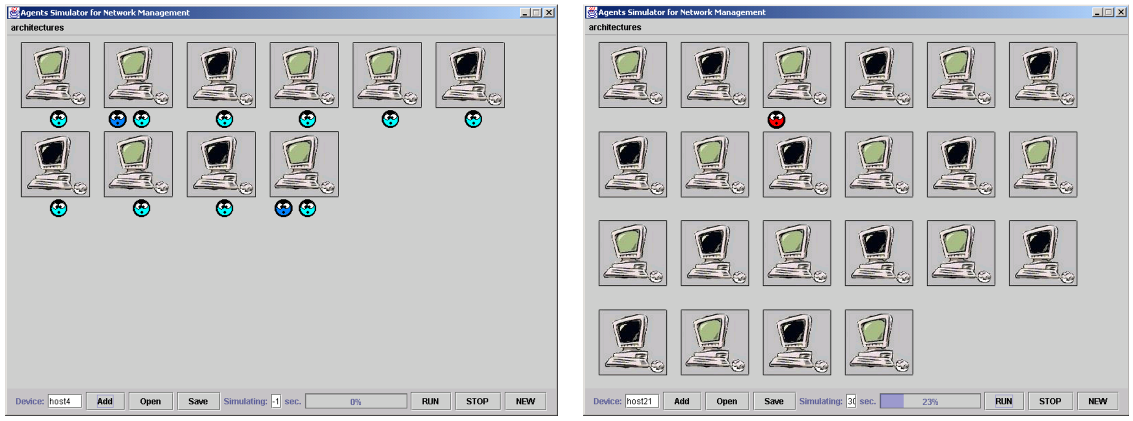


Fig. 7 – Two simulation environments. On the left: Several *Sentinels* and two *MobDecideMobDoit* agents; on the right: one *MobEntire* agent.

Calibration

In order to guarantee the reliability of the results of our experiments, we have calibrated the simulator using data collected from real tests on the network. In other words, we have implemented and run agents to perform each sub-task described in Section 4.3. Then we have measured the actual costs of each sub-task in terms of processing time and CPU consumption, etc. Finally, the measurements have been introduced in the simulator. Each sub-task or operations have been achieved or executed 100 times.

The sub-tasks have been implemented in three different ways: *local execution*, *remote execution through UNIX Remote Shell (rsh)* and *remote execution through CORBA remote objects* (see Table 2). The Java code lines bellow show how the rsh execution is handled for listing host partitions, listing files of a directory, getting file information, and removing a file, respectively.

```
Process p = Runtime.getRuntime().exec("ssh "+host+" /bin/df -kl");
Process p = Runtime.getRuntime().exec("ssh "+host+" /bin/ls -lAc " +path);
Process p = Runtime.getRuntime().exec("ssh "+host+" /bin/file " +file);
Process p = Runtime.getRuntime().exec("ssh "+host+" rm "+file);
```

The CORBA remote objects have been implemented using Borland Visibroker [18] with IDL compiler for Java. The costs of agents’ migration and agents’ inter-communication have also been measured (see Table 3). The cost is different for agents with knowledge base, i.e. capable of making decisions, and without knowledge base. An agent with a knowledge base is heavier than an agent that does not have a knowledge base. The mobility middleware used was ObjectSpace Voyager [19].

Sub-tasks	Local Execution		Remote Objects		Remote Shell	
	time (msec)	CPU	time (msec)	CPU	time (msec)	CPU
Partitions classification	9.0	1%	90.0	1%	399.0	1%
Partitions utilization checking	1.0	1%	110.0	1%	698.0	1%
Files classification	111.0	12%	325.0	15%	1896.0	14%
Actions choice	0.36	2%	13.3	9%	0.36	2%
Actions execution	125.0	3%	259.0	3%	160.0	3%

Table 2. Processing time and CPU consumption average for 3 different implementations of the sub-tasks

Migration				Communication			
KB		without KB		Unicast		Multicast	
Time	CPU	time (msec)	CPU	time (msec)	CPU	time (msec)	CPU
430	10%	120	6%	306.0	1%	145.0	4%

Table 3. Processing time and CPU consumption average for agents migration (KB = knowledge base) and communication interchange.

Operations	Voyager		CORBA	
	time (msec)	CPU	time (msec)	CPU
Platform initialization	5399.0	10%	2977.0	8%
Object Binding	-	-	2855.0	8%

Table 4. Platform's initialization and object binding

Such values are represented by static constants of the class `Parameters`. Fig. 8 shows the definitions of some of these constants. The first two ones refer, respectively, to the average time and the CPU consumption for the Voyager platform initialization. The other measurements are represented on the same way.

```
public class Parameters implements ISParameters {
   public static final int TIME_INIT_VOYAGER = 5399;
   public static final int CPU_INIT_VOYAGER  = 10;
   public static final int TIME_INIT_CORBA   = 2977;
   public static final int CPU_INIT_CORBA    = 8;
   ...
   public static final int TIME_CPART_OR     = 90;
   public static final int CPU_CPART_OR      = 1;
   ...
   public static final int TIME_CFILES_LOCAL = 111;
   public static final int CPU_CFILES_LOCAL  = 12;
   ...
   public static final int TIME_MULTICAST    = 145;
   public static final int CPU_MULTICAST     = 4;
   ...
}
```

Fig. 8 – Constants of the `Parameter` class

For understanding the accumulation process for these values during a simulation, suppose the simulation of the *MobMonoComplete* architecture with one *MobEntire* agent as illustrated in figure 7 (the right part). This example implies the following operations*: Voyager platform initialization, partitions classification, utilization checking, files classification, actions choice, actions execution, KB-agent migration*.

For each of such operations, a specific variable accumulates the total processing time. The pseudo-code below shows how this process is done:

```
initSimulation() {
   tot_init_voyager ← Parameters.TIME_INIT_VOYAGER
   Inicia thread de execução do agente MobEntire
   While the simulation is not finished:
      tot_cpart_local ← Parameters.TIME_CPART_LOCAL x NP
      tot_checkUtil_local ← Parameters.TIME_CHECKUTIL_LOCAL x NP
      For each partition with super-utilization problem
          Insert the partition in the KB
          For each file of the partition
              tot_cfiles_local ← Parameters.TIME_CFILES_LOCAL
              Insert the file in the KB
          Insert a call counter in the KB
          Insert a counter for decisions taken in the KB
          Execute KB
          tot_decision_local ← Parameters.TIME_DECISION_LOCAL x C
          tot_exec_local ← Parameters.TIME_EXEC_LOCAL x D
      Calls migration()
}

migration() {
   tot_KBmigration ← Parameters.TIME_KBMIGRATION
}

/* where,
   NP = number of partitions in the machine where it is executing
   tot_... = variable that accumulates the times defined in the class Parameters
   KB = Knowledge Base of the agent

*/
```

5. EXPERIMENTS AND RESULTS

We have performed many experiments with five different multi-agent organizations (*LocalMonoComplete*, *MobMonoComplete*, *ConnectMonoComplete*, *MobLocalMixed*, *MobTwoSpec*), 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 initially simulated a network with 30 machines without super-utilization problems to verify how well the machines have been visited. During the given experiment timespan, the organization *ConnectMonoComplete* (either in rsh or Remote Object implementation) have visited only 50% of the machines. This undesirable behavior is due to the high costs of frequently network access of *FarEntire* agents (agents that solve the whole problem remotely). On the other hand, the organizations *LocalMonoComplete* and *MobLocalMixed* have visited all machines more than the necessary times, since the Sentinel agents perform “visits” continuously (Fig. 9). The other two organizations have done a satisfactory number of visits. Fig. 9 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.

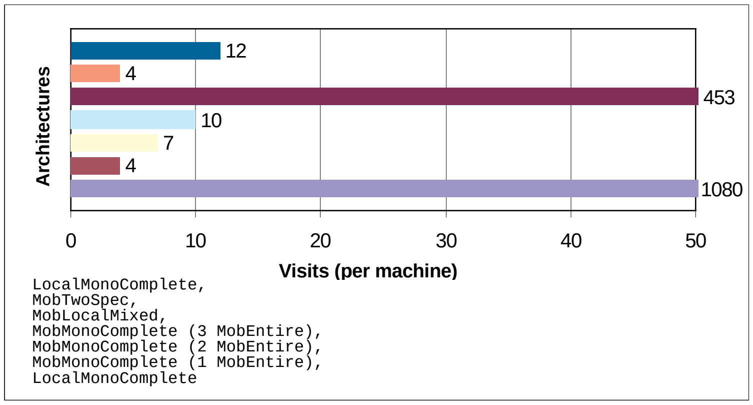


Fig. 9 - Number of visits in a 30-machine network without of super-utilization problems

In networks with low rates of super-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 on the network. Fig. 10 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 *MobEntire* agents in the organization *MobMonoComplete*, the processing time and the management latency (the average time super-utilized machines remain unmanaged) 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. 10 - 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 organizations with *MobEntire* agents such as the *MobMonoComplete* one, such agent has to process all the operations; there is no previous processing. However, the communication overhead is much larger in that then it is in this one: as soon as the Sentinels accomplishes 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 larger 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 get the calls in time. We may also notice 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 on the performance of an organization when we increase 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 many more migrations in the first case than in the last one (Fig. 11).

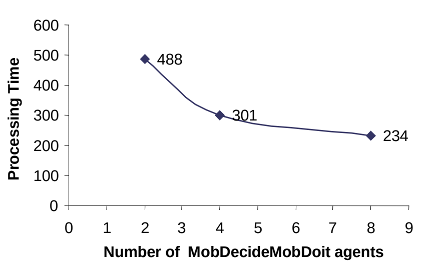
As an example of the influence of the mobility, the autonomy degree and the properties correlation, the results obtained with the *MobTwoSpec* organizations have shown that an increase in the number of *MobSentinel* agents causes a great increase in the number of migrations and in the number of multicast 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 the large communication 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. 11 - 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 results from the *saturation effect* in the management. That is, the 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 increased the number of *MobDecideMobDoit* agents. However, the improvement rate seems to tend to stabilization with the increase in the number of these agents (see graphs bellow).



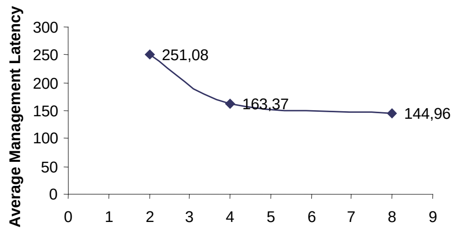


Fig. 12 - Stabilization trend for both processing time and average management latency with the increase of the number of *MobDecideMobDoit* agents.

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 a management latency on average larger than an organization with specialized agents (static or mobile ones) for monitoring, decision and execution. More details in [17].

6. CONCLUSIONS

In this work, we have made an original analysis of some agent's properties, in particular mobility, autonomy and distribution, for the network management field. Agents seem to be a promising approach. However a great amount of research effort must yet be done in order to conclude on how and when use this approach, as well as to obtain a precise evaluation of its potential advantages and limitations.

We are already working to adapt the agent types and organizations, as well as the simulator, to different management problems, in order to propose a specific methodology of agent-based approaches to network management. Considering the results we have had, we also intend to build a distributed agent-based system for disk space management. Finally we plan to include learning capabilities to the agent models, in order to achieve adaptive behavior in deciding when to visit a device and what to expect, in terms of pitfalls, in a given visit. This adaptation would provide various improvements, such as avoiding unnecessary visits.

REFERENCES

1 Harrison, C. G., Chess, D. M., Kershenbaum, A. (1996) Mobile agents: Are they a good idea? Technical report, IBM Research Division.

2 Berson, A. (1996) Client-Server Architecture, 2nd edition. McGraw-Hill.

3 Andrade, R. de C., Macedo, H. T., Ramalho, G. L., Ferraz, C. A. G. (2001) Distributed Mobile Autonomous Agents in Network Management. To appear in: Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001). Las Vegas, USA.

4 Bieszczad, A., Pagurek, B., White, T. (1998) Mobile Agents for Network Management. In: IEEE Communications Surveys.

5 Puliafito, A., Tomarchio, O. (1999) 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.

6 Russel, S. and Norvig, P.: Artificial Intelligence (1995) A Modern Approach. Prentice Hall.

7 Baldi, M., Gai, S., Picco, G. P. (1997) Exploiting Code Mobility in Decentralized and Flexible Network Management. In: Proceedings of Mobile Agents 97 (MA97).

8 Case, J., Fedor, M., Schoffstall, M. L., Davin, C. (1990) Simple Network Management Protocol (SNMP). RFC 1157.
9 McCloghrie, K., Rose, M. T. (1991) Management Information Base for Network Management of TCP/IP based internets, MIB-II. In: Internet Request for Comments Series RFC 1213.
10 Mitchell, T. (1997) Machine learning. McGraw-Hill.
11 Lesser, V. R. (1999) Cooperative Multi-agent Systems: A Personal View of the State of the Art. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1.
12 Bieszczad, A., Pagurek, B. (1998) Network Management Application-Oriented Taxonomy of Mobile Code. In: IEEE/IFIP Network Operations and Management Symposium (NOMS'98). New Orleans, Louisiana.
13 Minar, N., Kramer, K. H., Maes, P. (1999) Cooperating Mobile Agents for Mapping Networks. In: Proceedings of the First Hungarian National Conference on Agent Based Computation.
14 Puliafito, A., Tomarchio, O. (1999) 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.
15 Cheikhrouhou, M. M., Conti, P., Labetoulle, J. (1998) Intelligent Agents in Network Management a State-of-the-Art. Network and Information System Journal. Volume 1, no. 1, 9-38.
16 Figueira Filho, C., Ramalho, G. (2000) 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.
17 Macedo, H. T. (2001) Mobility, Autonomy and Distribution in Agents for the Management of Corporate Systems. Master's Thesis, CIn/UFPE.
18 Object Management Group (1998) The Common Object Request Broker: Architecture and Specification (CORBA), Framingham, MA.
19 Object Space Voyager. http://www.objectspace.com/products/voyager/
20 Silva, A., Ferraz, C., Ramalho, G. & Souza, J. (2001) Proactive Network Management Based on Mobile Agents and Fuzzy Logic. In Proceedings of the International Conference on Telecommunications - ICT'2001. Bucharest.
21 Hewlard Packard. Hp openview. http://www.hp.com/openview/index.html
22 Tivoli. http://www.tivoli.com/products/index/
23 Minar, N., Kramer, K. H. & Maes, P. (1999). Cooperating Mobile Agents for Mapping Networks. In *Proceedings of the First Hungarian National Conference on Agent Based Computation*.
24 Cheiknrouhou, M. M., Conti, P. & Labetoulle, J. (1998). Intelligent Agents in Network Management a State-of-the-Art. *Networking and Information Systems Journal*. Volume 1 – n° 1/1998, pp 9-38.
25 Rational Software Corporation. Unified Modeling Language (UML). http://www.rational.com