Journal of Network and Systems Management, Vol. 8, No. 3, 2000

A Software Agent Architecture for Network Management: Case Studies and Experience Gained

Morsy M. Cheikhrouhou,[1] Pierre Conti,[1] Karina Marcus,[1] and Jacques Labetoulle[1]

Current Network Management paradigms are rigid and lack flexibility. This makes the task of managing a highly evolving and dynamic network difficult to cope with. This paper presents the results of our work on Agent technology as a new paradigm for developing Network Management applications. First, we present our agent architecture that is built in a way that allows the agent to acquire new capabilities at runtime. Second, we present two case studies implemented with a prototype of this agent architecture. The first case study consists of an agent system in which faulty agents are automatically detected, their tasks then being reallocated to other agents, thus providing a fault-tolerant management system. The second case study deals with the configuration of heterogeneous ATM networks to establish end-to-end permanent virtual channels. Finally, we evaluate our agent architecture and the agent paradigm in general when applied to Network Management.

KEY WORDS: Skill-based agents; domain management; reliability; ATM management.

1. INTRODUCTION

We are observing an increasing interest in software agents in the context of Network Management (NM). This mainly appears through the standardization efforts of some agent aspects such as inter-agent communication [1] and mobility [2], the increasing number of research projects using agent technology in NM [3, 4] and the involvement of major industrial companies [5]. Classical NM paradigms, architectures and platforms currently face several challenges, such as the increasing complexity in size and heterogeneity, the highly rapid evolution of networking technologies, and the need for interoperability and cost reduction.

[1]Institut Eurécom, Corporate Communications Dept., BP 193, 2229 Route des Crêtes 06904, Sophia Antipolis Cedex-France. E-mail: {*Morsy.Cheikhrouhou, Pierre.Conti, Karina.Marcus, Jacques. Labetoulle*}*@eurecom.fr*

Proponents of the application of agent technology to the NM field maintain that agent properties [6, 7] are particularly suitable and have the potential to overcome the NM challenges. Agent *autonomy* allows the network administrator to delegate the most routine and time-consuming tasks to the agents, and to focus on high-level management tasks requiring human intelligence. *Social ability* and *communication* provide a better support for distribution and reduce the management traffic by exchanging only concise messages between the agents. *Reactivity* enables prompt reactions to be made to changes in network behavior, while *proactivity* enables these changes to be predicted, and preventive actions carried out to reduce their impact or to seize the opportunity for achieving high-priority management operations. The agent metaphor provides a high abstraction for handling NM problems and therefore both time and cost of application design [8] should be reduced. Finally, agents are particularly suitable for process control applications [9], which can be adequately adopted in the case of NM problems [10, 11].

However, in contrast to mobile agent applications, which already have begun to flourish, NM applications using the concept of static software agents are still limited in number. Whether this is due to the lack of standarization or to the overwhelming literature on agent theories and architectures, the result is that no long-term view has been taken of the real benefits of agent technology and of how problems have been successfully dealt with in the context of NM. Thus in this paper, our objective is to describe the work that we have done with software agents applied to network management.

The structure of the paper is as follows: Section 2 presents a rapid overview of the main trends, architectures, and principles of the agent paradigm. Section 3 describes the agent architecture that we designed and implemented as a full Java prototype. It also presents the inter-agent communication mechanism as well as the initial proposal of an agent development life-cycle. Section 4 presents our first case study, which describes how agents may improve the reliability of a NMS. Section 5 presents the second case study. Its purpose is to apply the agent prototype in the provision of Permanent Virtual Circuits (PVC) in heterogenous ATM networks. The approach is based on cooperative agents located on the ATM switches. The case studies are followed in Section 6 by a discussion that evaluates the added value of our agent framework and presents the major issues that we faced. Section 7 concludes the paper.

2. SOFTWARE AGENTS

It is admitted that the term “agent” has been excessively ascribed to many different concepts in several domains. However, it is easy to distinguish between two major trends that make use of the term. The first trend originated from works on code mobility which led to *mobile agents*, i.e., agents that are able to move from one host to another during their lifetime (e.g., [12–15]). The second trend originated from the domain of Distributed Artificial Intelligence (DAI), which focuses on computational entities which have some kind of intelligent behavior, and which may accomplish complex tasks by cooperating together. Such agents are often referred to as *intelligent agents*.

We are committed to the second trend, and focus the remainder of this paper on "(software) agents". We consider an agent as a "reactive [16] computational entity that is capable of autonomously accomplishing tasks and of communicating with other agents". There are excellent papers describing general aspects of agency [6, 7, 17]. What follows is a brief summary of the most relevant aspects of agent technology: architecture, communication, and coordination.

An important way to characterize agents issued from DAI is their internal architecture. The *deliberative architecture* relies on a logical model of the external world, mostly using mental notions such as beliefs and intentions [6, 18]. The *reactive architecture* does not reason on a model of the world and uses a stimulus-response type of programming instead. The *hybrid architecture* combines both architectures. While deliberative agents are capable of reasoning on the world's past, current and expected states to plan for long-term actions, reactive agents are suitable for building agents with prompt reactions. Hybrid agents benefit from the advantages of both kinds.

Another aspect to consider is inter-agent communication. A structured communication protocol can be used such as KQML (Knowledge Query and Manipulation Language [19, 20]) or the FIPA (Foundation for Intelligent Physical Agents) ACL (Agent Communication Language [1]). Such agent communication protocols are based on the speech act theory [21] that attributes intentions or *performatives* to the messages exchanged between the agents. Some communication languages include negotiation and cooperation mechanisms such as those included in COOL [22]. Other agent applications may use proprietary communication languages, but this option does not allow for interoperability between agents from different systems. Finally in some applications agents exchange information indirectly by modifying the status of the world, for example, using a shared external blackboard.

Cooperation is also an important aspect of agency. *Intelligent cooperation* means that agents are able to perform global planning and to infer cooperation scenario on the fly. In general, such agents are able to accomplish *explicit* global goals. In a less powerful cooperation mechanism, cooperation scenarios are hard-coded into agent behavior. It is also possible to conceive of an agent system in which agents do not interact at all, but this is not common in a distributed application. Finally, there are self-interested agents which do not cooperate. Instead, they act in a selfish way trying to maximize their own profit without any consideration for the overall agent system.

We believe that different NM applications can benefit from different agent cooperation models. Therefore, instead of committing to a particular type of agent, we favor a modular agent architecture in which agent cooperation can be designed to fit a particular NM problem. This will be presented in the following section.

3. OUR AGENT ARCHITECTURE

Nowadays NM systems must manage networks that use rapidly evolving networking technologies. Therefore they need to be able to adapt their behavior and functionality seamlessly whenever needed. This directed our design decision to an agent architecture that is composed of two parts. The first part is a common kernel that encodes the basic yet evolvable agent functionality. The *Brain* of the agent is the "headmaster" that governs its overall behavior. The second part is composed of *Capability Skill Modules*, which provide the agent with the specific and necessary capabilities to ensure particular management functions, which can be acquired at runtime. In addition, agents are endowed with a communication mechanism that allows them to communicate and cooperate in an easy and efficient way. A prototype of the proposed agent architecture was fully implemented in Java. We detail these elements in the next subsections.

3.1. Capability Skill Modules

A *Capability Skill Module* (or *skill* for short) is a set of Java classes that can be instantiated into the agent to provide it with a new management role or function. It defines new information elements called *Beliefs* and the necessary *Capabilities* that bring the required know-how.

3.1.1. Beliefs

The term *Belief* is borrowed from the Intelligent Agent research community and is defined by Müller [9] as "the agent's expectations about the current state of the world and about the likelihood of a course of action achieving certain effects". In the context of network management, agent beliefs hold the management information the agent has, as well as information about the other agents it knows about. For example, the status of a switch that the agent is managing can be expressed as a belief: `(switch :name sw301 :ipAddr 197.48.55.63 :status Ok)`.

Each skill must define the beliefs related to its management functionality. For example, a skill that manages ATM switches must define and instrument the necessary beliefs on the managed switches, including the statuses of the virtual paths and virtual channels created. The skill is responsible for ensuring the coherence and integrity of the beliefs it generates. In addition, it is the skill which defines the complete syntactic and semantic properties of its beliefs.

The beliefs brought by the skills are all made available to the agent's brain and are centralized in the *Belief Database* presented later in Section 3.2.

3.1.2. Capabilities

A *capability* is a type of action that the agent may invoke from a specific skill to achieve a certain management task or operation. Capabilities allow the agent to either interact with the managed network or perform higher-level management functions such as fault diagnosis or performance analysis. Capabilities that allow the agent to interact with the managed network can be either *sensors* or *effectors*. Sensors enable the status of the network to be captured by using a management protocol such as SNMP. Effectors enable the managed elements to be configured.

In general, a capability may invoke another capability within the same or another skill. In addition, a capability may involve either a long-term activity, like monitoring the job queue of a certain printer, or a function-like action, like adding a route entry on a certain router.

3.1.3 Skill Declaration

To allow the Brain to make use of a loaded skill, a declarative interface is provided within each skill. This interface allows the brain to discover the beliefs and capabilities offered by each skill. For the beliefs, only the syntax aspects including the name of the belief and its attributes are declared. And for each capability, the skill indicates the following:

- The preconditions that must be satisfied before it can be invoked.
- The other capabilities that it may invoke for its different execution instances. These prerequisite capabilities can be defined either in the same or in another skill.
- The beliefs that are used as input for its execution instances. These beliefs can be provided by other skills.
- The beliefs that are produced or updated during the achievement of the invoked capability.

3.2. The Brain

The Brain offers basic and innate facilities necessary for the agent operation. In Fig. 1, we distinguish the four functional parts of the agent's brain: the belief manager, the skill manager, the inter-agent communication mechanism, and the skill interface.

3.2.1. Belief Management

The *Belief Database* centralizes all the beliefs produced by the skills which have been loaded into the agent. In this way, beliefs can be accessed concurrently from any skill. The *Belief Manager* allows the skills to query and modify the

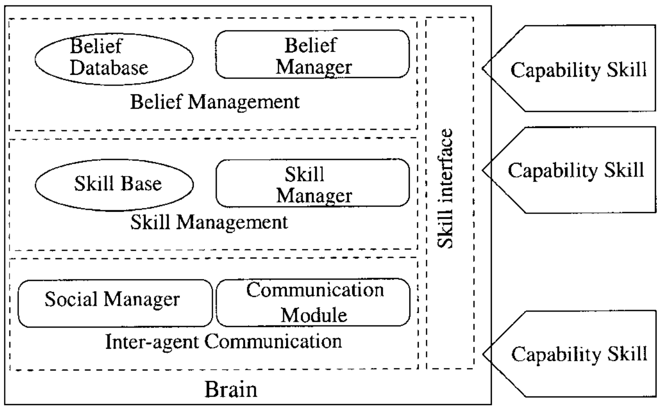

Fig. 1. The agent architecture.

agent's belief database. Besides the usual database operations, the belief manager allows the skills to "subscribe" as listeners to specific changes in the database. For example, a skill that is responsible for monitoring the printing service would subscribe to the creation and deletion of beliefs corresponding to printing devices (which in turn could be supplied by a discovery skill for example).

3.2.2. Skill Management

The *Skill Base* holds the declarative information that describes the beliefs and capabilities brought by each skill known to the agent. Using this knowledge the agent can make efficient use of the functionality offered by each skill, the *Skill Manager* determining which skill can provide a certain capability that is required for the achievement of a particular management task, and which beliefs result from its invocation.

The Skill Manager is also responsible for searching, loading, and unloading agent skills. If a local search fails, the Skill Manager sends search requests to the other agents. If an agent has the requested skill, it can serialize its code to the requesting agent.

Finally, the Skill Manager can automatically forward the relevant belief changes brought about by the execution of a requested action to a skill or to another agent.

3.2.3. Inter-Agent Communication

The Brain offers inter-agent communication facilities that allow skills from different agents to interact in a transparent way. Skills deal only with the symbolic names of the distant agents they want to interact with, and they are not aware of distribution-related details in the agent system.

The *Communication Module* is responsible for sending requests to and receiving requests from other agents; whereas the *Social Manager* holds information (records) about the other agents, such as the host on which they run as well as their network addresses. These records maintained by the Social Manager are mapped into beliefs in the belief database. These beliefs allow skills to make use of the knowledge acquired about the other agents. For example, they allow an agent registration skill to be built that makes it possible for an agent to insure the role of an agent directory server.

The Communication Module enables agent messages to be sent and received. Many network protocols can be supported including raw TCP, UDP, HTTP, and SMTP. Therefore, the communication network protocol can be chosen according to the size and the nature of the information to be exchanged between the agents. The agent communication primitives are detailed in Section 3.3.

3.2.4. The Skill Interface

The *Skill Interface* allows for the dynamic plugging of skills during the agent operation. It also ensures the important role of managing the skill-brain interaction. This interaction can be:

- belief queries which are forwarded to the Belief Manager, or
- task invocations and skill loading orders which are forwarded to the Skill Manager, or
- communication acts which are forwarded to the Communication Module.

The brain-to-skill interaction concerns either notifications for relevant belief changes or requests for skill declarative information.

3.3. Agent Communication

Agent messages are organized into performatives [19] that indicate the intention behind the content of the message. This kind of message format, which adopts Speech Act Theory [21] is widely used in the agent community. We defined a set of ten performatives:

- **Tell:** Allows an agent A to inform another agent B of a subset of A's beliefs.
- **Ask:** Allows an agent to query the beliefs of another agent. Replies are contained in `tell` messages.
- **Insert, delete, update:** Allows respectively the addion, removal, or modification of a belief or a set of beliefs in the receiver's belief database.
- **Subscribe:** Allows an agent A to inform another agent B that A requires to be notified of specific changes in B's belief database. In this case, B must send `tell` messages for each relevant change in its beliefs.

- **Unsubscribe:** Allows the effect of a preceding `subscribe` message to be stopped.
- **Achieve:** Allows the receiver to reach a certain state on the managed network. The content of an `achieve` message is an abstract goal that forces the agent to look for the necessary actions to achieve it.
- **Request Action:** Allows the execution of a specified action to be delegated to the receiver agent.
- **Stop Action:** Allows an action that has been launched due to a preceding `requestAction` message to be stopped.

As can be noticed, the first eight performatives concern beliefs, while the last two performatives concern capabilities. But the belief-related performatives may also lead the receiver agent to execute consequent actions.

3.4. Developing Agents

This section provides an overview of the agent development process that is proposed for our skill-based agent architecture. The process is divided into three phases: macro-design, micro-design and agent deployment.

3.4.1. Macro-Design

In this phase, the developer identifies the major agent roles needed for the system to be developed. Each role is a specialization of some aspect of the overall management system. We wish to emphasize that the roles of an agent can be changed dynamically during its lifetime according to the skills that the agent has at any given moment.

Once the agent roles are identified, the behavior of the agent system can be described using interaction scenarios between these roles.

3.4.2. Micro-Design

This phase is a refinement of the Macro-design. It focuses on the skills that implement the identified roles instead of the agents themselves. A set of skills must be designed to ensure each of the roles identified in the first phase. A new skill can be based on the beliefs and capabilities of other existing skills.

A skill is completely specified when all its beliefs and capabilities are fully defined. This specification includes the integrity constraints that must be ensured for the beliefs, and the possible relation of these beliefs with other beliefs from other skills. The specification also includes the relationship between each capability and its prerequisite beliefs and capabilities. The output of the micro-design allows the scenarios described in the first phase to be refined to the level of detail of the interaction between the skills themselves.

3.4.3. Implementation and Deployment

The skills can be written without paying attention to distribution-related issues (thanks to the brain inter-agent communication facilities). The problem of agent distribution among the available network hosts can be handled just after the skills are developed. At this stage, parameters such as CPU load balancing, and the placing of agents appropriately throughout the network can optimize response time and bandwidth usage. Furthermore, the attribution of skills can be handled and tailored during the operation of the agents.

3.5. Discussion

We had two design concepts for our agent architecture. The first was that agents have to be highly flexible so that their behavior could evolve dynamically at runtime. This is mandatory because current networks evolve at a rapid pace in their technology, topology, and supported services. For this reason, we built our architecture on dynamically pluggable skills, which are perfectly supported in Java.

The second design choice was to have no commitment to any particular agent technology. Instead, we favored an approach that offers the basic functionality of a NM oriented software agent, but that in addition can support supplementary agent techniques when needed. We think that agent technology as a whole is valuable in the NM field, but different NM functions may require completely different agent techniques. Accordingly, our agent followed a horizontally layered agent architecture [23] similar to that used by Skarmaeas and Clark [24]. A major advantage of this layered architecture is modularity. Moreover, the fact that horizontal layers, or skills in our case, have direct and concurrent access to the agent sensory information and effectory capabilities is a valuable support to the flexibility and extendibility required in state-of-the-art management paradigms.

KQML and FIPA ACL have largely inspired the adopted agent communication language we adopted. The main advantage of such languages is that they dissociate the content of the message from the intention behind it. This allows for enhanced interoperability and support of multiple languages for the message content. The implemented communication language differs from KQML in two aspects. First we chose to encode agent messages in a more concise format so that the generated traffic is reduced and messages are easily parsed. Second we adopted a different set of performatives more suitable for NM purposes. We included the `requestAction` and `stopAction` message types, which allow us to have direct control over the agent activity. This is very practical in NM and allows the direct mapping of task delegation between agents.

The next two sections detail the case studies implemented so far using this agent architecture. Both sections are organized according to the three phases of the proposed development process.

4. ENSURING THE RELIABILITY OF AUTONOMOUS DOMAIN AGENTS

4.1. Rationale

The problem of the reliability of management agents has never been a major issue in classical management paradigms. Classical agents did not have major management responsibilities and all the decisions were taken at the manager level. Also the management protocol used to communicate with these agents intrinsically ensured the reliability of the agent. For example, the SNMP protocol mainly uses confirmed communication for SET operations, or polling-based monitoring in which each GET operation expects a reply.

However, these factors are no longer ensured when a distributed NMS (Network Management System) based on highly autonomous agents is used, where agents are capable of taking high-level decisions and have the authority to execute sensitive management operations, without direct control from the network administrator. Furthermore, the agents themselves make use of the managed network, which is fault prone.

Therefore the agent-based NMS must be able to promptly detect the unreliability of the management agents. Furthermore, when an agent is detected to be unreliable, the other agents should be able to cooperate together in order to ensure the management tasks that were previously assigned to the unreliable agent. This requires agents to be capable of dynamically undertaking new management tasks during their operation.

The purpose of this first experiment is to build a prototype of an agent-based NMS in which intelligent agents, each of which is allocated to a distinct domain of the network, are able to mutually test each other and to detect the agents that become unreliable due to network or system failures. These domain intelligent agents are able to dynamically redistribute their management tasks in the case of an agent failure, so that all the tasks continue to be ensured.

As an example of a distributed management task, we choose the monitoring of network elements for fault detection purposes. Each agent is allocated to a network domain in which it performs the monitoring sensitive network components. To detect the unreliability of the autonomous agents a distributed diagnosis algorithm is used, which will be presented later in this section.

In the following we present the way we proceed to identify the required agent roles and correspondent skills. Each identified skill is then presented, with the intra/inter-agent interaction involved. The experimental results and conclusions about this case study close the section.

4.2. Role Identification

We propose a scenario in which human network administrator requests a monitoring of sensitive network elements, e.g., routers, hubs and switches, to detect faults that may occur. The administrator connects to a particular agent that displays the required data. The managed network is divided into domains, which are assigned to the agents. Each agent performs detailed monitoring of the network elements of its domain and comes up with the high-level status of each element, which is then reported to the agent to which the administrator is connected.

The monitoring and fault detection activities must be performed, *even* if one of the agents is faulty. To ensure that only reliable agents send the information, an SLD algorithm [25] is used.

A *Manager Agent* is designated to centralize the monitoring information, and other agents, called *Domain Agents*, wait for the Manager commands. The Manager is in charge of assigning domains to domain agents, to delegate the task of monitoring, and to verify the reliability of each agent before displaying the monitoring information. If one agent is found to be faulty, then the Manager will reassign the domains, so that the monitoring may reliably continue; if one unreliable agent recovers from its failure, then the Manager reassigns the domains to balance the global workload of the domain agents. The agents execute all these tasks through skills, which will be presented in the next section.

4.3. Skill Design

In the description of each skill the reader will find indications (e.g. ①) about the information flow between the skills. They refer to the arrow numbers in the diagram presented in Fig. 2. In this diagram we depict the hierarchical composition of our application, and detail in each agent the skills that are involved in the information exchange. Bold arrows represent inter-agent communication acts, while the thin arrows are simple brain notifications about belief changes.

The proposed implementation relies on six different skills: four that run in the manager agent and two in the domain agents. The Manager agent runs an Interface skill that provides a GUI through applets. The Agent Management skill is responsible for the domain, and the Fault Management skill uses this designation to delegate monitoring tasks. The SLDM (SLD for the Manager) skill is the centralized part of the SLD algorithm. It uses the SLD skill that runs on each domain agent and that uses in its turn the Monitoring skill. The following subsections detail these skills.

We wish to emphasize that being a Manager does not require a dedicated agent. Indeed, thanks to the possible dynamic loading of skills, any agent can potentially ensure the role of the Manager by loading the necessary skills. As a result, a domain agent may ensure the functions of the Manager agent.

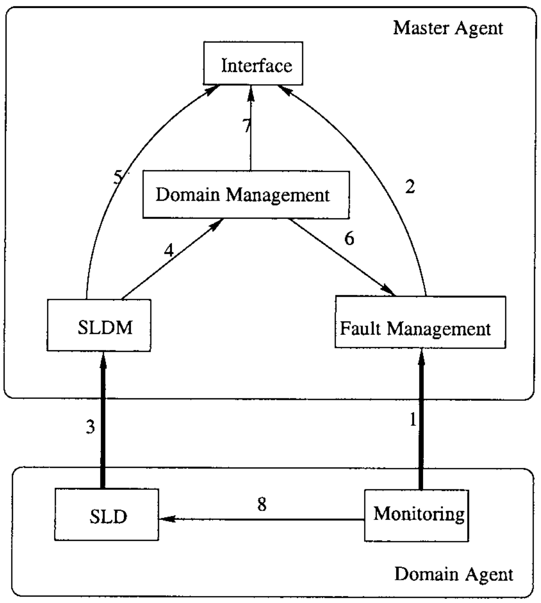

Fig. 2. Skill interaction diagram.

4.3.1. Domain Management Skill

This skill is loaded into the Manager Agent to make it responsible for domain management. The Domain Management skill is notified whenever a change in the beliefs about the actual set of domain agents occurs. This change may be produced either by introducing and removing agents dynamically or by a reliability analysis done using the SLD method ④. If this is the case, the Domain Management performs a domain redistribution in a way that ensures that all the requested management tasks are still performed. The domain allocations change, and this in turn provokes the brain into notifying the fault management skill ⑥ and the interface skill ⑦.

The following is an example of a belief corresponding to the domain allocation.

```
(Agent Domains :agent Ironman :domain (hub101 hub102 hub201)
               :agent Hulk :domain (sw101)
               :agent Batman :domain (hub301 hub303 sw202) )
```

4.3.2. Fault Management Skill

The Fault Management skill is loaded into the Manager agent. It is responsible for delegating the monitoring tasks to the domain agents. The delegation process is based on the domain allocation ensured by the Domain Management skill ⑥. The domain agents send the monitoring information ① to the Manager agent, which then verifies the reliability of the agent before making its information available to the Interface skill ②—that is, the agent name must be in the set of domain agents.

4.3.3. Monitoring Skill

The monitoring skill is responsible for the instrumentation of the necessary beliefs about the monitored hosts. This information is then used by the SLD skill ⑧ and the Fault Management skill ①.

The general scope of the monitoring skill is to monitor network element information in a protocol-independent way. It looks for the best way to instrument these parameters and decides the best way of getting this information (polling and polling frequency, traps, SNMP requests or RPC commands, etc.). The monitoring skill is able to produce a summary of the global status of a network element from a detailed monitoring.

4.3.4. System Level Diagnosis Skills

This skill module has been designed to allow a system of agents to ensure reliable behavior from each agent in the system, using a well-known method called system level diagnosis (SLD) [26, 27].

A Manager SLD skill (SLDM) is responsible for deciding which agents are reliable [25]. Every other domain agent executes a local SLD skill, which is in charge of the testing.

The SLD theory stipulates that a correct diagnosis about the agents can be established, whenever no more than half of the agents is faulty [28]. (We consider that this assumption is always fulfilled.)

Based on a test graph, the SLDM skill assigns to each domain agent a set of neighbor domain agents. To execute the testing, the SLD skill accesses beliefs produced by the monitoring skill ⑧ about some chosen network elements—the representative elements—in the agent domain and in the domains of its neighbors. The domain agents exchange the collected information about the representative elements and compare them to their own beliefs, in order to diagnose their neighbors. Afterwards, each SLD skill sends the test conclusions to the SLDM skill ③, which runs the diagnosis algorithm on the conclusions and updates the set of domain agents, eliminating the unreliable one(s). The Interface skill ⑤ and the Domain Management skill ④ are then notified of this modification.

4.3.5. Interface Skill

The interface is mainly developed for the demonstration. The network manager uses a Web browser to access the Manager agent, establishing goals and receiving results or notifications.

In this experiment the Interface skill provides:

- multiple access to the Manager agent from any standard Java-enabled browser
- the demands of agent predefined management goals
- the display of the Manager agent information on the execution of the goals
- network monitoring information per domain agent
- the status of each agent

The interface skill allows many applets to connect at the same time.

4.4. Deployment and Results

The local network of *Institut Eurécom* was used as a testbed for the experiment. It was decomposed into five domains, each of which was allocated to one domain agent. The Manager role was allocated to a different agent.

The experiment was composed of two stages. In the first stage, the monitoring was launched on the five domain agents that started to forward the results to the Manager agent. A domain agent called *Hulk* crashed. The crash was not detected by the agent system, and the monitoring of the network elements in Hulk's domain was no longer ensured. The results displayed on the GUI about Hulk's domain were out-of-date and even incorrect: one network element in Hulk's domain was switched off and the manager did not detect this.

In the second stage, the SLD mechanism was enabled. As before, the monitoring was launched on the domain agents and the applet started to display the monitoring results. Afterwards, the domain agent *Hulk* crashed. In this case, the crash was detected by the SLDM skill, which updated the beliefs describing the set of the available domain agents. The brain notified the domain management skill, which performed a domain redistribution. The network elements that had been previously monitored by *Hulk* were allocated to the other domain agents in a balanced way. So the network administrator was able to receive the monitoring information despite the failure of a part of the agent system. The system became fault tolerant.

Notice that the Monitoring, Fault Management and Domain Management skills offer the basic functionality of the system that was used in the first stage. The SLD skills were added later without causing any change in the basic modules. This shows the evolution and dynamic properties that the modularity of our agent framework offers to Network Management.

The next case study targets other aspects in NM including configuration and ATM management. It makes use of different properties of our agent framework.

5. PROVISION OF PERMANENT VIRTUAL CIRCUITS IN ATM NETWORKS

The second case study dealt with the configuration of Permanent Virtual Connections (PVC) in ATM networks. From an ATM operator stand point it is very useful to automate the provision of PVCs, which is a tedious task in heterogeneous ATM networks. The presentation of this case study requires an ATM background, which is outside the scope of this paper (see [29–31]).

5.1. Rationale

PVCs are unidirectional end-to-end connections that are manually established on a switch-by-switch basis. The establishment of a PVC is not a simple task. First, a physical end-to-end route between the source and the destination must be selected. Each node in the route is a triplet that identifies a switch and its selected input and output ports that are going to be used. There might be several physical routes and one of them must be chosen.

The next step is to plan through which Virtual Paths (VP) the PVC will be bundled. Several strategies can be adopted leading to different levels of optimization and implementation difficulties. The approach generally adopted by ATM operators consists in creating Virtual Paths between each consecutive switch on the route. Though it is far from being an optimal solution in terms of the number of supported VCCs, it is an easier solution.

Next one needs to attribute Virtual Channel Identifiers (VCI) on each switch and to create the necessary entries in the switch routing tables. On each switch, a VCI is needed for the input port and another for the output port. The output VCI of an intermediate switch must be the same as the input VCI of the next switch. In cases where local VPs are created on each switch to transport the PVC, the outgoing VPI (Virtual Path Identifier) of a switch must also be the same as the incoming VPI of the next switch. If one of these constraints is not satisfied, then data transmitted on this PVC will not reach its destination.

Finally, the different fragments of a PVC must be configured with the same Usage Parameter Control (UPC) contract. If a UPC parameter is wrongly configured at a switch, then the whole traffic on the PVC could be affected. Troubleshooting such problems is particularly hard.

But what can complicate the task of PVC creation even more is heterogeneity. Until now, every ATM fabric provider has elaborated its proper management interface with a proprietary configuration and administration interface. Moreover, if SNMP is supported, each provider uses a different MIB (Management Information Base). Therefore, the human network operator must know all these management interfaces to be able to appropriately create an end-to-end PVC.

In summary, the establishment of a PVC between end systems needs to consider a lot of parameters, and to satisfy some constraints that are hard to check, especially in a heterogeneous environment. Therefore, a management application that automates the provision of PVCs can really help ATM network operators in providing a more rapid service to their customers.

In the next section we describe the identified roles and the necessary skills for these roles, and specify their interactions. We then provide details on the implementation and deployment of the agent system.

5.2. Scenario and Agent Roles

We consider an ATM network operator offering end-to-end connections to its customers. Two distant users may want to establish a connection channel, for example to initiate a videoconference session. User Agents (UA) represent the network users, while Switch Agents (SA) that configure the network devices represent the network operator.

The role of the UA is to capture the user's requirements and the PVC parameters including the destination and the quality-of-service. The UA may automatically suggest the best suitable quality-of-service required for the type of connection that the user wants to establish. Once the parameters of the PVC are fully determined, the UA sends the request to the nearest SA on the operator side. If the PVC is successfully created, the UA is informed of the chosen PVC identifiers and may configure by itself the user equipment to make use of the new PVC.

The role of the SAs is to configure the switches in the ATM network. They accept the PVC creation requests from the UAs and cooperate together in order to satisfy these queries. Each switch in the ATM network has a dedicated and unique SA to configure it and to coordinate its configuration with that of the other switches. Any SA may potentially accept a request from a UA to establish a PVC. The SA that accepts the request for an end-to-end PVC is responsible *vis-a-vis* the demanding UA for the creation process of this PVC. We call such an SA, an *End-to-end Agent* regarding that particular PVC. The other SAs involved in the creation of that PVC are called *Local Agents*. The Local and the End-to-end roles are sub-roles of the Switch Agent's role. We emphasize that the same SA can be at the same time the End-to-end Agent for a particular PVC, and a Local Agent for another PVC.

5.3. Agent Skills

5.3.1. User Agent Skills

According to the description of its role, the UA behavior can be implemented using two skills. The *Contract Negotiation Skill* is responsible for sending PVC requests to the SA and negotiating the service contract as well as the price. The *User Interface Skill* is responsible for capturing user requests for a connection establishment and formulating them for the Contract Negotiation Skill.

The current implementation of this case study provides a simplified version of these two skills. The User Interface Skill is only composed of a Graphical User Interface that allows the user to specify the destination of the connection and the desired QoS contract. This request is then forwarded to the Contract Negotiation Skill which in turn delegates the task of PVC establishment to the SA, which manages the ATM switch that the user equipment is connected to.

5.3.2. Switch Agent Skills

The switch agent role is ensured by four skills: the *Switch Skill*, the *Local Skill*, the *End-to-end Skill*, and the *Topology Skill*.

5.3.2.1. The Switch Skill. A switch agent is responsible for PVC configuration operations of the switch it is allocated to. The *Switch Skill* is therefore designed to provide services to create and delete VPs and VCs using the SNMP management protocol. There is a switch skill for each different ATM equipment family. For example, the current implementation runs on FORE ATM switches, and therefore, a *FORE ATM Switch Skill* is developed. Actually, the switch skill is the unique part of the whole system that should be adapted for each product family.

In addition, the switch skill provides information related to the status of the current VPs and PVCs existing on the ATM switch. This information is used by the *Local Skill* to decide on the local parameters for establishing a new PVC.

5.3.2.2 The Local Skill. The *Local Skill* is responsible for the local configuration operations that create or delete a PVC fragment on the switch managed by the corresponding SA. For example, it is up to the local skill to decide whether to create a new local VP in order to convey the PVC within, or to use an already existing VP with sufficient available bandwidth. Also, it determines which VPI/VCI couples are to be assigned to the newly created VPs and PVCs.

5.3.2.3. The End-to-End Skill. The *End-to-End Skill* is responsible for the global supervision of the PVC establishment. Once a physical end-to-end route is found between the source and destination users, the end-to-end skill contacts the local switch agents on that route, and asks them to perform the necessary operations to create the PVC. The End-to-end Skill is also responsible for handling creation errors that might occur on switches.

5.3.2.4. The Topology Skill. Finally, the *Topology Skill* helps the end-to-end skill to identify a physical route between the source and the destination. The

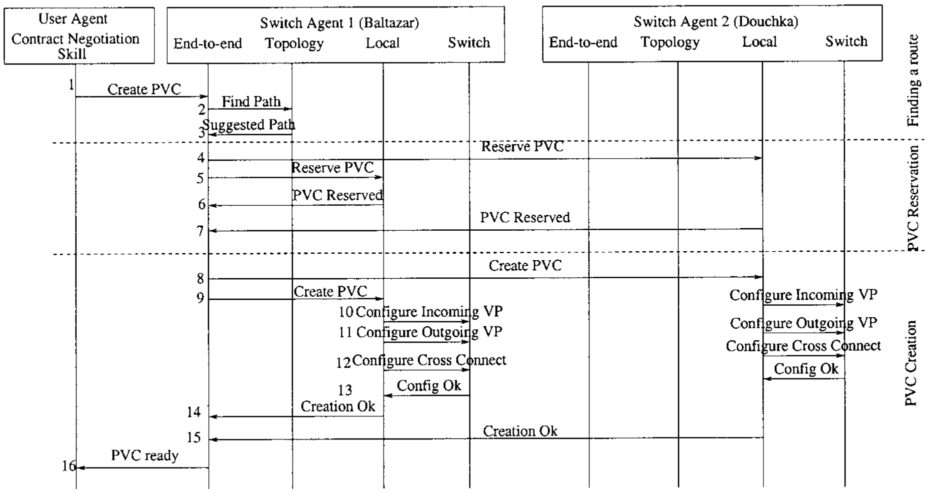

Fig. 3. PVC creation scenario.

physical route identifies which switches the PVC must use, and the input and the output ports that shall be used.

Finding a physical route is of a minor concern for us since we are more interested in the PVC configuration problem than with the routing issues. In our demo the topology of the network is hard-coded inside the topology skill source.

5.4. Results

The creation of an end-to-end PVC requires three major steps that are coordinated by the End-to-end Skill. These steps are detailed in Fig. 3 that shows the interactions between all the skills of a user agent (on the left) and two switch agents (respectively identified as "Baltazar" and "Douchka").

1. Finding a physical route. The end-to-end skill queries the topology skill for a physical route that links the source to the destination. The topology skill indicates which is the set of switches to be traversed and which input and output ports are to be used. The interaction between the end-to-end and topology skills is performed through the second and third messages in Fig. 3.
2. PVC reservation. In this phase, the end-to-end skill asks the local skills on the SA (including its own agent) to reserve the PVC (messages 4 and 5). Each local agent then checks whether it is possible to accept the PVC (messages 6 and 7). If it can be accepted, then all the parameters of the PVC are determined at this phase.

3. PVC creation. If the SA that is responsible for the global creation of the PVC receives positive acknowledgments from the other SAs, then the PVC can be effectively created. Again, the end-to-end skill sends creation commitments to the switch agents where the local skills execute the commitment using the services of the switch skill.

Finally, when all the positive acknowledgments for the PVC creation are received, the end-to-end skill can send back a message to the UA indicating that the PVC is now created and ready.

6. DISCUSSION

Though many people intuitively believe in the potential of software agents as a new paradigm to tackle NM problems, there are only a few case studies that concretely allow evaluating agent technology. The case studies presented allowed us to make some conclusions about the actual advantages of agent technology as well as to identify some weak points or challenges.

6.1. Flexibility

Flexibility is defined as the ease with which a system can be extended or adapted to new environments. From an NM point of view, it evaluates the degree to which the administrator is not limited by constraints imposed by a certain deployed technology. For example, typical SNMP agents are not flexible because the administrator is limited by the SNMP protocol primitives and by the degree of instrumentation offered by the agents.

Our developed agent architecture allows for an enhanced degree of flexibility. From an organizational standpoint, agents are initially considered as peer-to-peer entities. It is possible however to design an agent application with a different agent organization. The SLD case study is designed with a hierarchical organization, while the PVC case study is designed rather with a Client/Server paradigm. Push and Pull models can be used. In no case is the developer constrained to a specific paradigm or model, but is, instead, able to select the most efficient and suitable organization model.

Importantly, our agents can be developed so as to dynamically select the best interaction scenario according to the actual behavior of the network. Agent roles can also be easily changed, evolved, and re-assigned during the operation of the agent system. In the SLD case study for example, the role of the Manager Agent could be ensured by one of the Domain Agents.

As a general rule, any agent architecture or framework targeted to NM should allow for the dynamic change and evolution of the agent behavior. Java technology helps to support these features through the dynamic loading of classes that can encapsulate new behavior patterns.

6.2. Dynamism

The capability of our agent architecture to handle dynamic aspects is proved on several occasions in the case studies. At the agent level, it is easy to introduce or remove agents from the agent system. In the SLD case study, a new domain can be re-assigned simply by modifying the corresponding belief provided by the Domain Management Skill. For the PVC provision case study, adding a new switch asks only for the introduction of a new Switch Agent into the system. Except for the physical routing part (which was not considered in the case study, see the Topology Skill in Section 5.3), there is no change required for the Switch Agents.

Our agent architecture allows also for the dynamic attribution and change of agent roles. This only requires the plugging of the necessary skills that implement the new role.

6.3. Distribution and Efficiency

The agent paradigm inherently allows gains to be made from distribution. Agents deployed close to network elements take advantage of the processing capabilities of the hosts on which they are installed. In the PVC configuration case study, Switch Agents are allowed both to run reservation and configuration operations in parallel, and to perform all the SNMP interactions locally to the switches. These two factors reduce significantly the response time for PVC creation compared to a centralized approach in which tasks are performed serially, and the SNMP interactions are carried out remotely from a central NMS.

Another factor that allows for an enhanced performance is the high-level interaction between the Switch Agents. The messages exchanged between Switch Agents are concise and expressed in a high-level way. This leads to reduced traffic generated for the PVC creation compared to a centralized approach in which a large number of SNMP packets must be exchanged with every ATM switch in order to parse the routing tables and to set the PVC parameters.

6.4. Delegation and Cooperation

Delegation is an important concept both in NM and in software agents. In NM, delegation implies distributing management responsibilities amongst autonomous entities. Software agents inherently support the dynamic delegation of management tasks. This appears clearly in the first case study in which management responsibilities are assigned to agents dynamically during their operation. Moreover, it is possible to change this assignment without interruption.

In the second case study delegation was used to improve the efficiency and to facilitate the design of the management application.

Delegation is only a single aspect of agent cooperation. Another aspect is how the agents are organized among themselves. In both case studies there is a coordinator agent. A peer-to-peer approach could be used but is more difficult to design (from a distribution viewpoint) than using a central coordinator. However, we note that in both case studies, the central coordination organization was not hard-coded. In the SLD case study it is possible to select another agent at runtime as a Manager by loading the necessary skills. The PVC case study presents a dynamic coordination, because the role of the coordinator (i.e., the end-to-end agent) is only related to a particular PVC, and the same agent can insure at the same time the end-to-end role for a certain PVC and the local role for a different PVC. In this case, the burden of a central management station is avoided in favor of a cooperative approach.

Negotiation is another aspect of cooperation that was only partially considered in the case studies. In the PVC provision case study the User Agent may negotiate the PVC parameters, Quality of Service and price with the End-to-end Switch Agent. A negotiation skill may be developed and deployed without modifying the previous skills.

6.5. Protocol Independence and Support for Heterogeneity

The software agents handle management information using beliefs. Belief instrumentation is ensured by sensor capabilities independent of management protocols. For example, in the PVC provision case study, the Switch Skill provides the necessary capabilities for managing a specific type of ATM switch, but a different ATM switch may require a new Switch Skill. However, at the belief level, the agent has a unified view of all types of switches.

Also communication between the agents is achieved through communication acts that provide a high-level mechanism to exchange management information and tasks. In this way, agent communication does not depend on any network management protocol.

6.6. Agent Programming

A major problem when designing an agent-based NM application is the lack of awareness of agent paradigms and the immaturity of agent technologies. An agent developer is faced with a tremendous amount of agent theories, frameworks, and languages, and so design decisions are very difficult to make. This often means that the developer has a restricted view of the agent concept.

The lack of a clear design methodology tends to lengthen the development phase. For this reason, we proposed a development process that helped to develop the two case studies. The improvement of this process could be the subject of future work.

6.7. Agent Control

Controlling large distributed applications is by itself a complex problem. The control of an agent-based application is amplified by the nondeterministic behavior of the agents and their deliberative capabilities. This is a major challenge to the agent paradigm especially when agents are supposed to assume important responsibilities, which is the case in NM. Testability and proof of good behavior is also a major issue that motivated the development of our first case study.

In our case, to help deal with the problem of controllability, we provided a built-in Graphical User Interface for each agent. This interface allowed us to watch and control the agent operation by accessing its belief database, and to load and inhibit skills. In addition, both cases were developed using mainly reactive programming. Every agent generated a detailed trace of all the activities performed, which helped to debug the SLD and PVC agent applications.

7. CONCLUSIONS

The contribution of our paper is threefold. First, it presented an agent framework based on the notion of "pluggable" capability skills that allow the agent's brain to seamlessly integrate new capabilities and pieces of information. A prototype of this framework was fully implemented using the Java programming language. The second contribution consists in presenting two case studies tackling different aspects of NM. Through these case studies we were able to evaluate on a concrete basis the real advantages and challenges of our agent framework and the agent paradigm in general, which is the third contribution of our paper.

Our future work will deal with both the architectural and application aspects of our agent framework. On the one hand, we are studying how to include intelligent features such as planning and learning in our agent framework, and how to provide tools to ease the development of agent skills. On the other hand, we are looking for new case studies, which provide the best way to test and improve the concepts included in our agent prototype.

We believe that further similar studies will soon emerge and highlight other interesting aspects of static software agents applied to the network management domain. Such concrete applications of the agent paradigm are urgently needed to bring this new technology to a mature phase and to help elaborate agent standards.

REFERENCES

1. Agent communication language. *http://www.fipa.org/spec/fipa9712.pdf*, 1999.
2. Mobile Agent Facility formal specification, Object Management Group TC Document, cf/00-01-02, (January, 2000).
3. A. L. Hayzelden and J. Bigham, Agent technology in communications systems: An overview, *Knowledge Engineering Review*, Vol. 14, No. 4, pp. 341–375, 1999.
4. M. Cheikhrouhou, P. Conti, R. T. Oliveria, and J. Labetoulle, Intelligent agents in network management, a state of the art, *Networking and Information Systems*, Vol. 1, No. 1, pp. 9–38, 1998.

5. S. Albayrak and F. J. Garijo, (eds.), *Proceedings of the Second International Workshop on Intelligent Agents for Telecommunications Applications, IATA'98*, No. 1699 in *Lecture Notes in Artificial Intelligence*, Berlin, Springer, June 1998.
6. M. Wooldridge and N. R. Jennings, Intelligent agents: Theory and practice, *Knowledge Engineering Review*, Vol. 10, No. 2, pp. 115–152, 1995.
7. H. S. Nwana, Software agents: An overview, *Knowledge Engineering Review*, Vol. 11, pp. 205–244, October/November 1996.
8. M. Plu, Software technologies for building agent-based systems in telecommunications networks. In N. R. Jennings and M. J. Wooldrige, (eds.), *Agent Technology: Foundations, Applications and Markets*, Springer-Verlag, 1998.
9. J. P. Müller, *The Design of Intelligent Agents—A Layered Approach*, LNAI State-of-the-Art Survey, Berlin, Germany, Springer, 1996.
10. R. Oliveira and J. Labetoulle, Intelligent agents: A new management style, in *Proceedings of the Distributed Systems and Operations Management Workshop—DSOM'96*, L'Aquila, Italy, October 1996.
11. M. M. Cheikhrouhou, BDI-oriented agents for network management, in *Proceedings of Globecom'99*, Rio de Janeiro, Brazil, IEEE, December 1999.
12. T. Magedanz, On the impact of intelligent agents concepts on future telecommunication environments, in *Third International Conference on Intelligence in Broadband Services and Networks*, Crete, Greece, Octobr 1995.
13. C. Bäumer and T. Magedanz, Grasshopper—a mobile agent platform for active telecommunication networks, In S. Albayrak, (ed.), *Intelligent Agents for Telecommunication Applications* No. 1699 in *Lecture Notes in Artificial Intelligence*, Stockholm, Sweden, Springer, pp. 19–32, August 1999.
14. T. White, B. Pagurek, and A. Bieszczad, Network modeling for management applications using intelligent mobile agents, *Journal of Network and Systems Management*, Vol. 7, No. 3, pp. 295–321, September 1999.
15. P. Bellavista, A. Corradi, and C. Stefanelli, An open secure mobile agent framework for systems management, *Journal of Network and Systems Management*, Vol. 7, pp. 323–339, September 1999.
16. M. Wooldrige, Barriers to the industrial take-up of agent technology, in *The Practical Application of Intelligent Agents and Multi-Agent Technology*, London, p. 11, The Practical Application Company Ltd, April 1999.
17. G. Weiss, (ed.) *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, Massachussetts: The MIT Press, 1999.
18. M. P. Singh, A. S. Rao, and M. P. Georgeff, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press, pp. 331–376, 1998.
19. Y. Labrou and T. Finin, A proposal for a new KQML specification. Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, Maryland, February 1997. *http://www.cs.umbc.edu/kqml/papers/*.
20. T. Finin and G. Wiederhold, An overview of KQML: A knowledge query and manipulation language, 1991. Available through the Stanford University Computer Science Dept.
21. J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, 1970.
22. M. Barbuceanu and M. S. Fox, The design of a coordination language for multi-agent systems, *Intelligent Agents III. Agent Theories, Architectures, and Languages*, Springer, pp. 341–355, 1996.
23. M. Wooldridge, Intelligent agents. In G. Weiss, (ed.), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, Cambridge, Massachusetts, The MIT Press, 1999.
24. N. Skarmaeas and K. L. Clark, Process oriented programming for agent-based network manage-

ment, in *ECAI96 Workshop on Intelligent Agents for Telecommunication Applications (IATA96)*, Budapest, Hungary, August 1996.
25. J. Meinkohn and S. Albayrak, Future IN-platforms with agent technology. In S. Albayrak and F. J. Garijo (eds.), *Lecture Notes in Artificial Intelligence*, Springer p. 80, June 1998.
26. G. Berthet, *Extension and Application of System-level Diagnosis Theory for Distributed Fault Management in Communication Networks*. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, Lausanne, CH, 1996.
27. S. Hakimi and K. Nakajima, On adaptive system diagnosis, *IEEE Transactions on Computers*, Vol. C-33, pp. 234–240, March 1984.
28. F. P. Preparata, G. Metze, and R. T. Chien, On the connection assignment problem of diagnosable systems, *IEEE Transactions on Electronic Computers*, Vol. EC-16, pp. 848–854, December 1967.
29. M. M. Cheikhrouhou, P. O. Conti, and J. Labetoulle, Flexible software agents for the automatic provision of PVCs in ATM networks, in *Section IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, Helsinki, Finland, 1999.
30. B. Pagurek, Y. Li, A. Bieszczad, and G. Susilo, Network configuration management in heterogeneous ATM environments. In S. Albayrak and F. J. Garijo, (eds.) *Intelligent Agents for Telecommunication Applications—IATA'98* No. 1437 in *Lecture Notes in Artificial Intelligence*, Paris, France, July 1998.
31. K. Tesink and T. Brunner, (Re)Configuration of ATM virtual connections with SNMP, *The Simple Times*, Vol. 3, August 1994.

Morsy M. Cheikhrouhou received his Engineering Diploma in Computer Science in June 1996 from the École Nationale des Sciences de l'Informatique, Tunis. He joined the Corporate Communications Department of the Institut Eurécom in April 1997, where he started his Ph.D. thesis. His research activities are focused on Network Management, Intelligent and Mobile Agents, Java technology and distributed systems. His homepage is at: *http://www.eurecom.fr/~cheikhro*

Pierre Olivier Conti is a research engineer with more than 17 years of software development, QA management, project management and consulting experience in the Telecoms and Network Management areas. He mainly worked for Digital Equipment Corporation within the Telecom Engineering Group. He joined Institut Eurécom in 1997 to lead a research project on Intelligent Agents for Network Management, and to pursue a Ph.D. thesis in parallel.

Karina Marcus received her Ph.D. in Operations Research from the Université Joseph Fourier, France, in 1996. She then spent a post-doctoral year in Canada, and in September 1997 she joined the Institut Eurécom. She has published several papers in distinct areas such as matroid theory, parallel algorithms network design and fault detection for distributed systems using intelligent agents.

Jacques Labetoulle obtained his Ph.D. from the Université Paris VI (1974) and a "Doctorat d'Etat" from the Université Paris IX Orsay (1978), in computer science. He joined the "Institut National de Recherche en Informatique et Automatique" in 1970 where he mainly worked on queuing theory and applications. In 1981, he joined the CNET (Centre National d'Etudes des Télécommunications) to lead the teletraffic department. In 1989, he created the new center of CNET in Sophia Antipolis, where he headed the Corporate Network Management department. He joined the Institut Eurécom in 1992 to lead the Corporate Communications Department and the network management research group.