Intelligent Agents : a way to reduce the gap between applications and networks

Raul Oliveira, Jacques Labetoulle
Corporate Communications Department
Institute Eurecom
06904 SOPHIA ANTIPOLIS CEDEX, France.

Abstract

Network Management (NM) has been in some way an activity apart from any other carried on the networks. Monitoring networks and systems resources independently from how users and applications are using them, has proven to be an inadequate approach since in most of the situations it's impossible to trace the relationship between network or systems faults and the applications affected by them. Furthermore, the degradation of network and systems resources can be considered as a fault from an application point of view, and there aren't any form to be aware of this degradation/fault, since applications requirements are not known to the network management system (NMS), and even when it happens the network can apparently still be viewed as functioning. Currently, the management efforts are used mostly to diagnose the origin of the faults, or to notify the systems administrators of the presence of a fault which in our opinion could be perceived by the network manager at very late stage.

In this article we propose a way to bring the management closer to application and user requirements. The approach relies on spreading intelligent agents through network segments or domains, these agents being the management front-ends responsible to: collect QoS requirements concerning application operation over the network, monitor required resources, perform when possible corrective actions, or at least provide management staff with powerful data so that they can take the corrective actions faster than before. We expect to benefit from being aware of how network and systems are used to develop a proactive management in real time, without the burden of classical approaches.

1 Introduction

Over the years the fact that network management systems (NMS) hadn't been efficient enough to prevent faults, or able to alert critical applications of them, have conduced developers to insert fault detection functionalities in several applications running in industrial environments. In our opinion this results on effort duplication and spreading of vital information concerning the same problem over different and non connected entities, without any advantage from both sides. As a consequence, applications still don't have complete data about faults, whether in real time or not, and also they can't do anything more than prevent catastrophic behaviors based on information they have obtained. From the NMS side the problem is in some way similar, because without application information concerning network inability to accomplish its tasks properly it couldn't react before the problems or symptoms become faults that require human intervention.

This article claims that all activities of fault detection, diagnosis and corrective actions should be completely delegated to the NMS. To make feasible this approach the NMS obviously needs some more information than which it has at disposal currently. An application, as a network user, must believe that the network and associated services will always operate reliably and consequently doesn't need to perform any special effort to detect faults. However, this constitutes in some way an undefined target, depending on too many factors. Hence, the application should provide the NMS with precise requirements concerning operations over the network, from which the NMS should be able to offer the desired reliability.

Having an application believing that the network never fails is a comfort from the application point of view but isn't a pragmatic approach, so we think that it should be an NMS obligation to notify its users when the network is not anymore able to ensure the negotiated quality of service. As a last possibility, which "theoretically" should not happen, the application that haven't received an NMS alarm but also doesn't have a service responding appropriately, can issue a complaint about the network delivered quality of service.

The proposed framework entails several questions to an NMS such as: precocious diagnosis of faults, individualized diagnosis accordingly to application critical needs, and autonomous decision over which corrective actions should be taken in real time.

Current NMS doesn't have any mechanism that permits such an approach, they are not prepared to perform precocious diagnosis since they are mainly designed to distinguish between healthy or faulty services, which is not enough. NMS should be endowed with capabilities to deal with the lack of information between these two states, upon which it will be possible to detect some symptoms of oncoming faults.

Until now the NMS don't have any mechanism to specialize observations dynamically accordingly to what is really important in the network environment. This specialization depends obviously upon current network "users" and their critical requirements. Thus the NMS must also be able to receive application's requirements from which it will determine the subset of services/subsystems that should be monitored, as well as to send alarms to the appropriate users in case of inability to maintain a correct behavior of network services. Finally, and by far the most complex, the proactive side of this framework, requires the NMS to be able to decide which corrective actions to trigger to bring the network services to their optimum state.

Most of the issues raised here, even if we admited they could be configured more or less in nowadays NMS, will lead invariably to a high consumption of NMS resources and a non negligible traffic overhead. The number of services and associated subsystems in the network environment is extremely high, and one must remember that NMS network bandwidth utilization under every circumstance should be reduced especially during major network faults.

The current approach in our opinion will permit to reduce the gap between applications or users and the NMS, making network management an interactive and indispensable tool, on which a critical environment will strongly depends. Talking about interactiveness, who never gave a phone call to his network manager asking what is happening? We extended the scope of this weak notion of interactivity to a broader target, which seemed to us to be of major importance: provide NMS with knowledge about the users and their requirements.

Before going through the solution that we propose for this framework, we will review the state of the art in some domains closely related to our work and on which it will depend. For instance, our objectives are clearly situated under the umbrella of network management automation. And in this field one major aspect should be analyzed such as the role of AI in NM automation. Then in section 3 we discuss how should looks like an adapted management architecture. In section 4 we'll present what is for us an Intelligent e Agent (IA) and how it will be integrated in the overall management architecture. We conclude in section 5 where an overview of the IA architecture is presented. Finally we go through the conclusions, starting with a possible extension of this framework to traditional management environments, and finishing by resuming our expectations of what should be in the future the role of an IA.

2 Network management automation

Network management automation have always been a complex task for several reasons. For instance, network configurations are not standard at all, changing whether dependently upon types of activities or from environment to environment. Even inside the same corporation, network configuration evolution can be faster than what one always expected at their installation. The distribution of applications and their interactions with customer or network servers is even worse from this point of view. These might be seen as the main reasons to justify, until now, the nonexistence of tools, upon which it will be possible to build easily network management automation solutions. Thus the achievement of real time network management is a difficult target, although it's one of the most important ones, especially for industrial environments.

Some steps must be done to simplify the scenario in order to pursuit in this direction. One hypothesis that seems to be evident to us, is that the lack of information about what is being used in the network oblige an autonomous system to process huge quantities of data.

It's well known that a user application requires only that a subset of all the subsystems will be working correctly among those existing in the network environment. Thus maybe it doesn't make sense to demand to an NMS to monitor everything blindly.

Setting up traps or notifications upon some agent variables values having over-passed a determined threshold, requires clearly considerably less effort than have a precise view of actual and previous states of critical subsystems. Thinking that, to prevent faults on subsystems in a reliable fashion, we might need base linings of their evolution, clearly show us that the charge of the NMS will be untenable. This leads to the idea that the network and systems management could be only concerned, for fine grain monitoring, on what is actually very important to the applications (users). This approach however doesn't exclude the monitoring of other resources on the network that aren't very critical at the moment. The idea is just to concentrate the efforts for fine grain monitoring in what is really necessary currently, and from where the faults have higher probability to come from (user point of view).

Network management automation concerns, three major tasks: monitoring, analyze and reaction (take actions). From the beginning, most of the work done in network and systems management have been directed to the instrumentation and communications point of view. Although this is very important, for network maintainers the most important is to develop strategies that permit them to provide users with a network service which is the most reliable as possible.

Following the same reasons that we presented before it's clearly impossible to design network management applications from factory that will be able to successfully: monitor the appropriate data, analyze these data accordingly to environment characteristics and take the correct actions to fix up the problems.

The classical approach in commercial management platforms is the provision of facilities allowing the operator to define specialized data queries. This is usually done to set up alarms, but the fault diagnosis and the corrective actions must be taken by a human operator.

For a decentralized approach Management by Delegation (MBD) concept [14] and elastic servers [7] enables one to transfer management programs to an elastic server, being these programs executed on behalf of a manager. These programs or scripts composed of instructions carrying data queries, expression calculations and actions to take upon the results of the previous analyses. Since these programs could be transfered at any time it's possible to configure the network management activities according to the current needs and network configuration. Although the delegation of programs to elastic servers it done aiming at decentralizing management and avoiding micro management from a centralized manager station, it also proves itself to be very useful towards network management automation. Nevertheless MBD depends heavily on network manager expertise to write the programs to delegate, and so it doesn't address per si the real time network management requirements concerning namely applications in industrial environments.

One target that still is not to much explored is the network management automation towards real time, so our research will be mostly concerned with real time issues for NMS's.

Artificial Intelligence support

Network management automation is typically the shift of acquired experience from network managers to autonomous software applications. In doing it one expects to reduce the reaction times for well known problems to which the corrective actions are also known.

The challenge is to develop solutions that could trace problems effectively, without mistake, in now a days complex network environments. Most of the services offered on a network have high dependence on several subsystems being too hard to know precisely from where a problem comes from. A problem detected by a user or application on a service might not come from the subsystem implementing the service, but from another upon which the later depends. So the network expertise isn't anymore mapped on a first order heuristic rule and clearly needs some more powerful inference mechanism to analyze and follow the dependencies to find the trouble. Furthermore, it clearly needs that we feed up the intelligent systems with enough knowledge about network current configuration, services dependencies on subsystems, interdependencies between the subsystems and, as we claim, applications dependencies on services.

At first glance expert systems would be the "el dorado", but its utilization is still mostly appropriated to off-line operation. In fact most of these systems are still today standalone, working in isolation and independently of other systems performing, often, related tasks [8]. Besides other criticism that we could explore, such handicaps are already enough to reject this kind of solutions, for environments where real time and cooperation are strong requirements.

Several knowledge representations techniques exist today that might be applied to network management, especially in fault management field, ranging from : case based reasoning (CBR), model based reasoning (MBR), neural networks (NN) and Bayesian belief networks (BBN).

CBR is well suited for cases where a high number of situations described by relations between symptoms and faults exist, and where the environment isn't to much dynamic so that set of cases become obsolete. The startup of such an approach could suffer from the lacking of cases. MBR supports decision making based on a formal problem model. This approach permits starting the operation from the beginning with models constructed from design data. The question is if the models accurately captures the network behavior. If this is the case the operation of such an approach proceeds by finding discrepancies between the modeled and observed behavior. NN and BBN are what might be called probabilistic AI technologies and are appropriate for alarm correlation since they can analyze patterns of common behavior on networks, and can handle ambiguity and incomplete data [9].

Machine learning accordingly to some authors [8], is an issue to consider for future systems in order to keep pace with accelerating change in network technology. In our opinion this approach is the most complex one and should be avoided while other possibilities in the AI domain are explored. Most expertise in network management still non assimilated and the sources of knowledge are not yet all explored. Explore these sources should be the first goal to achieve before going towards machine learning.

In this article we try to explore one of these sources. Users and applications can play in fact a helpful role in providing knowledge[1] and hints to intelligent systems to develop their activity, much in the same way it happens

[1]Knowledge here means how to adapt standard management procedures to satisfy application requirements

nowadays between network managers and users, with the difference that this done in real time will be much more efficient. In our opinion, however, the real question remains unanswered: do we really need artificial intelligence (AI) techniques to build intelligent agents (autonomous software processes)?

3 Management architecture

The management architecture that we propose to achieve our goal of integrating users requirements in the network management, and extend the level of automation, requires a new type of entity with a dedicated role for that purpose.

In fact managing applications requirements from the point of view of quality of service appears to be an impossible task with the existing architecture of network management for several reasons :

- the managers are not provided with pertinent information about application requirements,
- even with this information, the managers are located too far from network resources and agents, and thus the process of managing individual requirements of applications may result in an excessive traffic load,
- dealing with this process may in any case result in an overload of work for managers,
- the traditional agents cannot be modified to treat the problem since they are designed in a very general fashion and cannot be updated for the purpose of a specific need.

It is thus necessary to create new concepts in term of architecture and functionalities to deal with this problem. Let us recall briefly the requirements this process implies:

- each (or carefully selected because of critical constraints) application must provide precise information about its individual requirements to the Network Management System,
- the NMS should be able to execute specific observations (measurements, testings, ...) derived from the application's requirements, run adapted algorithms to try to forecast and correct eventual future problems and notify the managers and/or applications if the process is not successful.

To be efficient, it is mandatory that the diagnosis is very fast and takes place before the problem may appear at the application level. This implies that the machine running this process is close to the equipments (and thus to the agents).

These processes can be considered in fact from the network users point of view as management front ends. As well, from the manager level they will be seen as autonomous "agents" with delegated rights to perform network management in an autonomous way, according to established policies. As we justify in the section 4 management front ends will be called Intelligent Agents (IA).

The proposed architecture (described in figure 1) tries to provide a realistic solution, by adding the following elements to traditional management architecture:

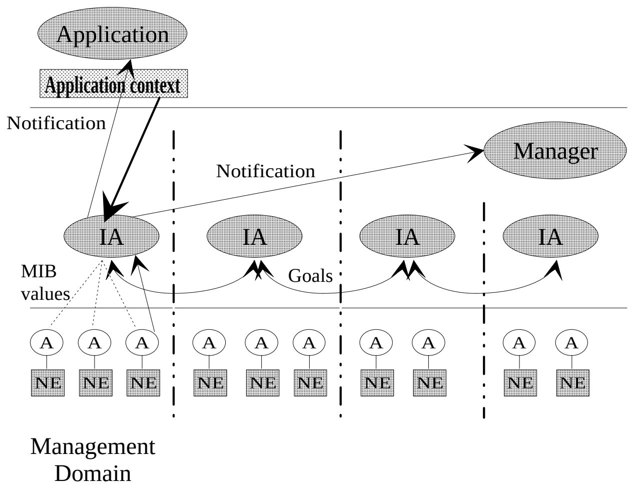

Figure 1:Enhanced management architecture.

- an information, called the application context, is added to the application. The context contains all pertinent information about the application requirements, i.e. identification of critical resources or services and the quality of service the application is waiting from them. This information can be internal to the application, if the application has been designed using the concepts of our architecture (this will be possible for future applications). If not, the context can be added to existing applications and executed just before the application is launched. The context, when executed, sends a set of information to a specific Intelligent Agent (IA) using an appropriate Network Interface (defined in terms of an API).
- any management area is provided with an IA, whose architecture is described in figure 2. The first component of the IA is a Context Manager who translates the application requirements in terms of goals the IA must achieve for the purpose of managing individual QoS. The IA is also equipped with an internal engine executing the tasks to satisfy the goals. Executing these tasks may result in : performing specific observations on existing MIBs [11] [3] (when corresponding

agents are located in the same management area), performing specific observations on programmable MIBs (typically RMON [12] MIBs), launching tests on resources (i.e. by sending periodically test requests to critical resources or services), defining and sending goals that other IAs must perform. The IA is thus provided with two interfaces (one for emission, one for reception) with all the other IAs.

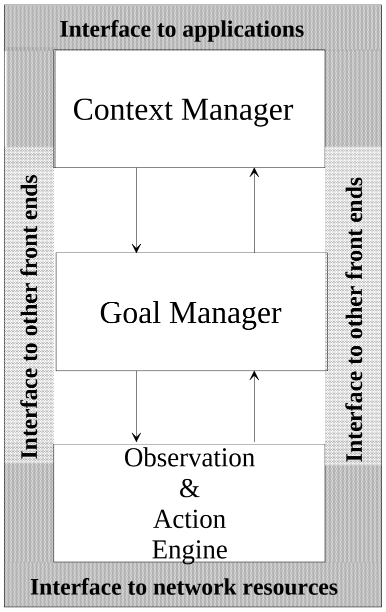
Interface to applications
Context Manager
Goal Manager
Observation & Action Engine
Interface to other front ends
Interface to other front ends
Interface to network resources

Figure 2:Intelligent Agent (management front-end) general architecture.

It's not clear until now how these agents can be distributed over a network environment. There are several approaches to consider for a domain, namely: groups of NEs, segments, subnets, addressing space or any other approach that reflects some type of organization within the global network environment. In the same way the hosts for these entities aren't also determined currently. Dependently of network configuration one can choose several possibilities like for example: routers, switches, hubs, bridges, dedicated hosts, network dedicated managers (field bus managers), etc.

Figure 3 presents possible distributions of IAs (backbone of our NMS), over a networked environment.

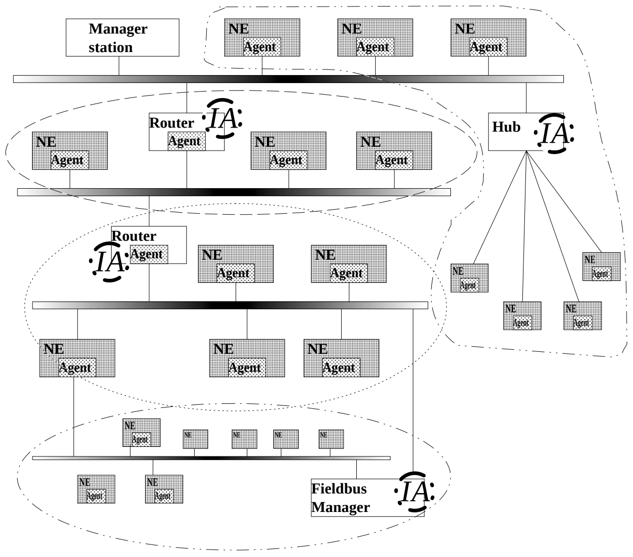

Figure 3:Intelligent Agents spread over a typical management environment.

4 Intelligent Agents

Before starting to describe what should be the role of an IA in the network environment the best is to clarify why do we have chosen IA instead of other name for a management process at an intermediate level, between managers and agents. The term *Agent* for the network management community is associated with a server process that offer instrumentation over resources at one network element (NE), to be consulted by a management process.

However, a more general view of the agent concept would point us to a broader definition. Hence, the agent concept enjoys the following properties [13]:

- **autonomy** - agents operate without the direct of humans or others, and have some kind of control over their actions and internal state.
- **social ability** - agents interact with other agents (and possibly humans);
- **reactivity** - agents perceive their environment (which may be the network environment, network segment,

domain, etc) and respond in a timely fashion to changes that occur in it;

- **pro-activeness** - agents do not simply act in response to their environment, they are able to exhibit goal directed behavior by taking the initiative.

This type of agent is conceptually different from NMS passive agents and so we decide to differentiate them from the later ones by calling them Intelligent.

As we claimed already, NMS in general suffer from the inability of humans operators to process the huge quantity of data available in current network environments. So the delegation of some of their tasks to other entities seems to be a natural step. It's easy to understand that the entities that are in charge to substitute human beings should have the properties mentioned above.

The reason to demand to the network users to inform the NMS of their real requirements, is in some way very logical from the AI point of view. We should remember that most of the work on AI has to do with the problem of finding ways to attack search problems, such as these with limited computational resources available in practice. So instead of searching blindly, we should use some sort of heuristics, or rules of thumb, to focus our efforts [6].

Application context

User requirements are delivered to an IA (NMS) through application contexts in the same way as the ACSE [1] definition for the creation of peer to peer associations. Here the application can either negotiate its context or just inform the IA. The first approach is clearly the best since it allows the network to have some kind of control over shared resources, trying to maintain acceptable levels of utilization. For instance, for Ethernet networks this will constitute a starting point to implement policies to reduce traffic on connections over-passing the "negotiated" traffic, in order to avoid congestion situations.

Applications contexts are the means through which user applications specify their requirements. Figure 4 gives an idea of what these contexts may include.

- *Application name*
- *Application priority*
- *List of services*
  - *list of QoS parameters for the service*
    - *Request rate*
    - *Utilization (of global service capacity)*
    - *Service time*
    - *Error rate*
    - *Throughput*
    - *Response time*
    - ...
  - *Service fault severity*
  - *Resources offering the service*
  - *Redundant resources for the service*
  - ...
- *List of global QoS parameters*
- ...

Figure 4: Example of possible application context attributes.

The application contexts can be either very detailed or not, depending on the level of standardization of services and resources mentioned in the context. So we admit that in most cases an IA can find in the overall NMS the knowledge to test or diagnose the service providers or resources. For non standard cases the context should also include this knowledge or point to where this knowledge can be found. For instance we can suppose that a user application is built on top of non standard services, especially designed for the current application. In that case the IA will be pointed to where it can find information upon service decomposition and individualized test procedures for services or resources. For example, to test network resources, like temperature sensors on a field bus (non instrumented NE's), the IA will be pointed to the appropriate testing procedure.

Since application's contexts can become very complex, we opted to divide the global application context in several sub contexts. With smaller contexts it will be easier for applications to pass their requirements to the NMS, each application accordingly to its needs can pass either simpler or very detailed contexts. Figure 5 shows how the contexts are organized.

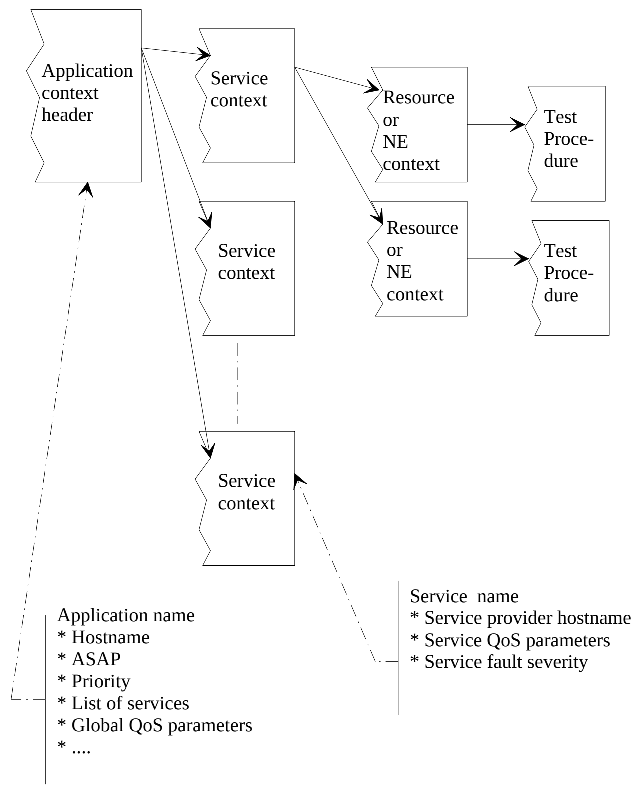

Figure 5:Application context organization in sub contexts.

The global context separation in several sub contexts enables the application to update its context during run time without need to do it for the overall context. This also reduces the number of active goals in the IA. For instance, one application can just update the context header and in this way inactivate some of the service oriented sub contexts.

Intelligent Agent goals

IA in spite of their "intelligence" must not disperse their monitoring efforts, through all data available over the network (in MIBs located on the NEs). This is the major problem of a static approaches where without dynamic selection criterion the data monitoring activity will have to preview all possible interesting scenarios. In our approach the "heuristics or rules of thumb" to select these scenarios are in fact coming from the network users requirements. Based on this information the IA must determine where to focus the monitoring effort.

As mentioned above the agents exhibit goal directed behavior, and in our case, their goals are constructed dynamically according to user needs. Each time an application enters the network environment, the IA will have a new set of goals. Dependently of network organization in domains some of the created goals could entail the cooperation of others IAs. Thus these goals must be forwarded to the appropriate IA. Nevertheless, delegated goals responsibility still belong to the goal creator. For instance, if the cooperating IA isn't able to guaranty the delegated goal, he must notify the goal owner, which will notify the applications concerned.

The idea of controlling agents behavior, based on goals derived from contexts, is not only supported by agents theory, but also has as advantage to create an independence between goal semantics and operations that the IA will perform to verify goal achievement. The homogeneity of the goals yields a separation of user requirements from distinct information models and heterogeneous management definitions, for resources that have identical properties as viewed by users. In fact, each IA on his domain of responsibility could have different operations sets to achieve similar goals, dependently on available instrumentation at the involved managed nodes. Otherwise, if a goal is to be sent to another domain, is up to the local IA to decide by which means he will verify the received goal. This option will prove to be more efficient and open, since even IAs based on distinct technologies can still cooperate.

Communication issues

The proposed architecture besides the introduction of two additional elements to traditional management architectures, also creates two new axes of communication, nonexistent until now, between:

- applications and IAs (middle level management process)
- IAs themselves, to share goals, goals results and knowledge

The former communication axe is clearly the simpler one, and we can propose naturally that the applications will use existent protocols available for network management, such as CMIP [2] or SNMP [4]. This will simplify the design of IAs since they already need to use these protocols to access either agents on NE or the manager station. The approach sends the problem to the formalization of management objects that will be able to carry the contexts. The challenge is to design these objects in a way that they could host all possible contexts required by applications. From figure 4 and figure 5, we can say that this doesn't seem to be the most complex task.

For the communication axe between the IAs, neither of the communication paradigms commonly used in network management currently seems appropriate. In fact the IAs will need to share goals among them, demand management procedures to verify goals and rules to fix problems from identified faults. These information types aren't adapted to the structure of the management information (SMI) used by management protocols such as CMIP and SNMP. Intelligent Agents research community have been dealing for long time now with these problems, and have enough experience on Agents Communication Languages (ACL). An example of an ACL is KQML (Knowledge Query and Manipulation Language) [5] which uses KIF (Knowledge Interchange Format) expressions appropriate to this type of communication. We don't want to compromise ourselves with a choice right now, but KQML is a strong possibility for this communication axe.

5 The Intelligent Agent architecture

Research around the IA architecture is still being developed. Right now it's clear that an IA as an autonomous entity able to perform their tasks alone will lead, of course, to a complex architecture. We identify several major functional blocks for IA architecture that actually seems us to be of major importance to achieve ours targets.

Figure 6 gives an overview of the IA architecture which is divided in three main areas.

Contexts manager

Contexts manager (CM) is responsible for maintaining a real-time DB of application's contexts. This DB aims at storing instances of the different contexts types, as well as their relationships either among themselves or with the running applications on IA domain. CM is also in charge of goals creation from the received application's contexts, and the forwarding of events to applications and management stations when goals (associated with application's contexts) hadn't been successfully guaranteed (probably expressed with distinct syntaxes).

Goals manager

The goals manager (GM) is the heart of an IA, since its name will be mainly justified by this function. He has to process the received goals either from CM or remote IAs, and to survey their status so that the concerned entities (applications and remote IAs) could be notified. If the goals cannot be monitored locally they are forwarded to other IAs better positioned to survey them. GM and CM must share a real-time DB where are stored the relationships between goals and contexts entries, so when notified of an unsuccessful goal the CM could determine which context parameter is affected.

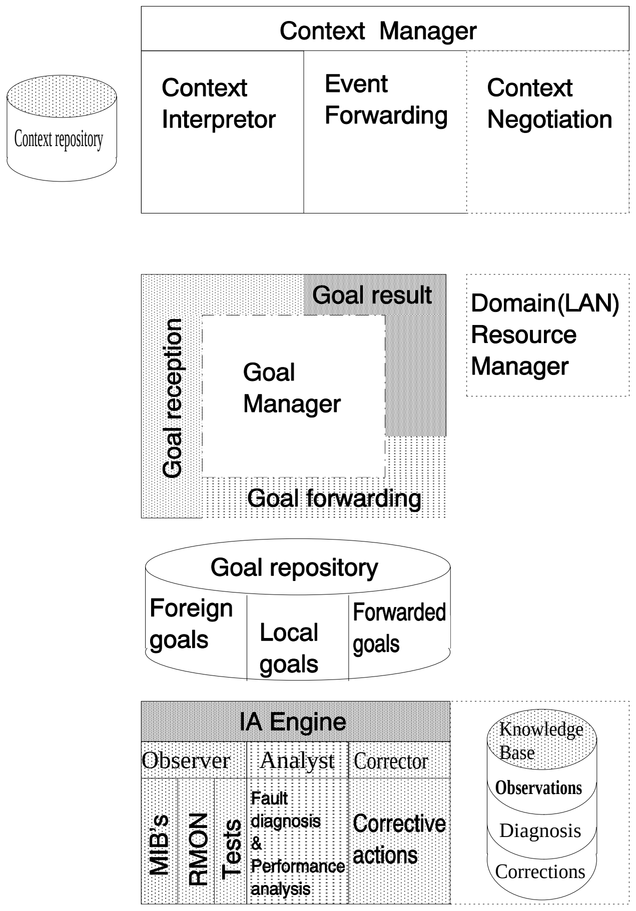

Figure 6:Intelligent Agent internal architecture.

Engine

The engine is organized in three functional blocks:

- **Observer** The observer is the process in charge of monitoring the network resources in order to verify goals accomplishment. Several types of observations are possible: monitoring MIB values, monitoring values on probe MIBs (configured remotely by the observer), launching tests on network resources or service providers, and reading systems logs in NEs. Since observations are made according to goals, when one of them isn't anymore achieved the observer informs the analyst of which goal is unsuccessful. Afterwards it is up to the analyst to start a diagnosis process.

- **Analyst** is the process in charge to perform fault diagnosis and performance analysis upon observer requests. The analyst at least knows exactly from where starts the diagnosis, because it knows the failed goal and its relation with services, subsystems or resources.

In case of success the analyst will provide the corrector with the identified fault, so that the later could try to fix it.

- **Corrector** is the process in charge to recover from faults. Typically having a precise identification of the faulty subsystem or resource, the corrector will try to bring it to the normal state. The procedures to fix the problems must be available internally on some repository or KB. We should notice that several services in LANs are dependent on remote processes for which a simple reboot action, in most of the cases, is enough to bring the service to normal state.

Both processes (corrector and analyst) in case of insuccess are required to update goal status, which will give place to the forwarding of event notifications to concerned applications and as well as to the manager station. We have to refer that the IA is a management process, and consequently all alarms (traps in SNMP terminology) generated by agents should be sent to the IA instead of to a manager station. So the analyst besides observation results he also has as processing input triggers the alarms generated by agents.

The IA architecture intends also to solve some of the problems a manager station is usually faced up, and which constitutes an information bottleneck. Alarms handling is one of these problems, first because the manager will receive alarms coming from several domains, concerning faults either on NEs or services. It's in fact to much to a human being to process manually all these alarms, but also for an automatic centralized process it isn't an easy task accordingly to the multiple sources of alarms and their non evident relationships and side effects between them.

In our approach a manager station instead of receiving raw alarms or event notifications carrying dedicated attributes (CMIP) or variables bindings (SNMP), will preferably receive notifications about failed goals. This constitutes a clear enhancement for the message semantic received by a network manager.

Knowledge sources

The IA must be endowed with enough knowledge to translate goals in management operations over managed objects available on MIBs. Since the goals are independent from any particular design of managed objects, there might exist several mappings for the same goal. The engine must have also mappings for information not included in standardized managed objects but obtainable from other sources.

It's clear from what we already saw for the IA architecture, that it will be necessary to have some kind of knowledge bases (KB's) to store information such as goals mappings on observation procedures, services modeling and their dependencies to fault diagnosis, and faults to corrective actions relationships.

6 Conclusion

The proposed framework could be seen as very complex from the eyes of inadvertent readers. In fact what we presented in this article was mostly directed to critical applications environments such as the industrial one. But we have to remark that the same framework can be used for traditional management environments. In this case a manager station can provide management contexts, instead of applications contexts, which have to be handled by IAs based on the same principles that these we had exposed in this article.

The gap between applications and the network operation is in fact big. For some people this seems to be a reason to delay the introduction of networks in their environments, for others this is an headache when they have to develop reliable applications for critical industrial processes. This article has presented a novel approach for network and systems management, in which user requirements for critical applications can be taken in account. Furthermore, the NMS can notify the applications of its insuccess to achieve their needs. We have proposed an enhanced management architecture that accepts user requirements through application's contexts sent to Intelligent Agents running in their domain of operation. The approach entailed news axes of communication for the management architecture, whose characteristics and solutions were analyzed. The article finishes with an overview of the IA architecture and its main building blocks.

References

[1] ”Association Control Service Element for Open Systems Interconnection for CCITT Applications," Recommendation X.217, CCITT, 1988.

[2] ”Management Information Protocol Specification - Common Management Information Protocol, ISO/IEC 9596-1, ITU X.711.

[3] ”Structure of Management Information - Part 1: Management Information Model,” IS 10165-1, ISO/IEC, May 1992.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin ”A Simple Network Management Protocol (SNMP),” RFC 1157, May 1990.

[5] T. Finin, G. Wiederhold ”An overview of KQML: A Knowledge query and manipulation language,” Available through the Standford University Computer Science Dept., 1991.

[6] Matt Ginsberg, " Essentials of artificial Intelligence," *Morgan Kaufman Publishers*, 1993.

[7] G. Goldszmidt, "Distributed systems management via elastic servers," *Third International Symposium on Integrated Network Management*, pp. 95-107, 1993.

[8] Shri K. Goyal, "Network and Distributed Systems Management," *Addison-Wesley Publishing Company*, Ch 21, pp. 539-577, 1994.

[9] Denise W. Gürer, Irfan Khan, Richard Ogier, " An artificial Intelligence Approach to Network Fault Management," *SRI International Technical Report*, 1994.

[10] David Hutchison, Geoff Coulson, Andrew Campbell, and Gordon S. Blair, "Network and Distributed Systems Management," *Addison-Wesley Publishing Company*, Ch 11, pp. 273-302, 1994.

[11] K. McCloghrie, M. Rose "Management Information Base for Network Management TCP/IP-based Internets," RFC 1213, IAB, 1991.

[12] S. Waldbusser "Remote Network Monitoring MI Base," RFC 1271, 1991.

[13] Michael Wooldridge, Nicholas R. Jennings "Intelligent Agents: Theory and Practice," *To appear in Knowledge Engineering Review 10(2), 1995*

[14] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network management by delegation," *Second International Symposium on Integrated Network Management*, pp. 95-107, 1991.