Customized network management based on applications requirements

Raul Oliveira, Dominique Sidou, Jacques Labetoulle
Corporate Communications Department
Eurécom Institute
06904 SOPHIA ANTIPOLIS CEDEX, France.
email: {oliveira | sidou | labetoul}@eurecom.fr

Abstract

To manage current network environments we have to change the existent network and systems management paradigms, if we even can admit that one exists. Network Management should not be restricted to protocols, MIB definition, and associated instrumentation. In the end, what is really important is to satisfy user application requirements. One should note that blindly monitoring raw MIB data values has not proven to be very useful to fix day to day network management problems. The reason is seen to be that the gap between raw MIB data values and root causes of problems is in most cases too important to tackle. Therefore, in order to drastically reduce the amount of root causes analysis processing, some kind of heuristics must be introduced. This paper proposes a framework based on the integration of user application requirements into the NMS process. From the architectural point of view, the framework is based on intermediate management systems – the so-called Intelligent Agents – which serve as management front-ends to user applications.

Keywords : Network management automation, Applications requirements, Management architecture, Intelligent Agents

1 Introduction

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 is 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.

The traditional approach entailed other consequences namely for environments populated of critical applications. Hence, over the years the fact that network management systems (NMS) had not 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, especially for those running in industrial environments. In our opinion, this results in duplication of effort and spreading of vital information concerning the same problems over different and non connected processes, without any advantage from both sides. As a consequence, applications still do not 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, the NMS is not able to 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 the one it has at disposal currently. An application, as well as a network user, must believe that the network and associated services will always operate reliably and consequently does not need to perform any special effort to detect faults. Hence, applications and users must provide the NMS with precise requirements concerning their operation and behavior over the network, from which the NMS should be able to maintain the desired reliability.

Having an application believing that the network never fails is a comfort from the application point of view but is not 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 required quality of service. As a last possibility, which "theoretically" should not happen, the applications that have not received an NMS alarm but also does not have a service responding appropriately, can issue a complain about the unsatisfactory quality of service delivered by the network environment.

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

Precocious diagnosis

Current NMS does not 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.

Flexible management

NMS needs to be flexible, since management can not be performed based on static procedures. Just because the networking environment is inherently dynamic: users and applications are entering the system unpredictively with different data, performance and service requirements. So an NMS must have mechanisms 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.

Proactiveness

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.

Problems with NMSs

Most of the issues raised here, even if we admitted they could be done more or less in nowadays NMS (i.e. without application intervention), 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 proposed approach in our opinion should 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 could strongly rely on. The exploration of the proactive style of management will be in focus as a very strong requirement, but it must integrated in a framework where the reactive and the interactive are still present. We must be pragmatic and assume that still are several cases which need the network manager presence. Others where the reactive are the fastest to recover from faults and reduces the unavailability time.

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. For instance, our objectives are clearly situated under the umbrella of network management automation. Then, in section 3 we discuss how should look like an adapted management architecture. In section 4, we will present what is for us an Intelligent 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, finishing with further issues.

2 Network management automation

Browsing MIBs is currently what most NMS are able to do. However, determining when the monitored values are within acceptable ranges is not so easy. For network managers this will be a nightmare, and is completely out of scope, so some kind of automation is therefore needed to shift computational intensive repetitive tasks.

MBD shifting management processing

One important step to shift computational intensive tasks was done with Management by Delegation (MBD) concept [13] and elastic servers [5]. Enabling one to transfer management programs to an elastic server, these programs being 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 is 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, since MBD rely on off-line programming it is difficult to deal with dynamic network environment characteristics.

Automation difficulties

Network management automation is therefore 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. Furthermore the creation of workgroups in network environments is starting to be a reality, and we need to capture also this type of organization since it brings very helpful information concerning traffic patterns inside these workgroups and between them. These among others 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.

When talking about network management automation it is straightforward to think about discovery tools, which are to some extent a good example of automation. But discovering network elements is much more easier than discovering workgroups, running applications topologies and conversations profiles. This seems to us to be a very important information that must be feed to NMS in order to be able to establish a realistic framework for network and systems management automation.

It is well known that a user application requires only that a subset of all the subsystems (service providers) to work correctly among those existing in the network environment. Thus maybe it does not make sense to demand to an NMS to monitor everything blindly. 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 does not 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.

Accordingly to the reasons we presented so far it is 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.

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 networks and systems taking into account applications requirements from quality of service point of view 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 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 the traditional management architecture:

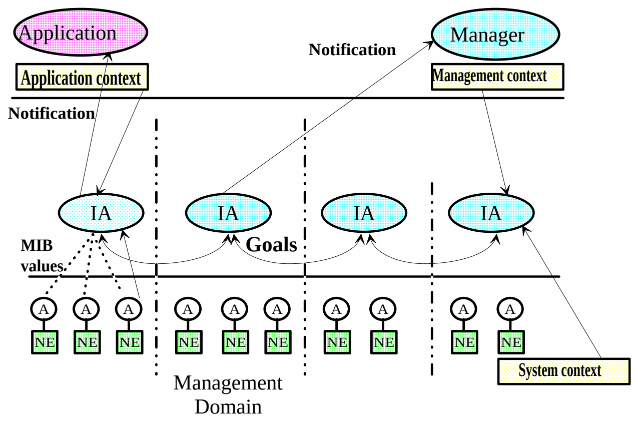

Figure 1:Enhanced management architecture.

- An information data structure, 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 it relies on and the quality of service (QoS) profile the application is waiting from them.

- An information structure called system context, which is provided by each host system. Systems contexts contains information about applications acting as service providers. These contexts types identify the resources upon which services providers relies on, as well as the QoS these services are able to offer to customer (e.g. client) applications.

- Each management domain is provided with an IA, whose layered architecture is described in figure 2. The first component of the IA is a Context Manager that translates the application requirements in terms of goals[1] the IA must achieve. 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 [7] [8] (when corresponding agents are located in the same management domain), performing specific observations on programmable MIBs (typically RMON [11] 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.

- An information structure called management context which must be provided by management applications controlled by human operators (manager stations). This structure is provided for main reasons: substitute applications and services contexts from legacy entities; specify high level management goals and policies so the IAs can have a scoped role to avoid overall conflicting behaviors, which might arise from an automated management environment.

[1]declarative statements about what has to be achieved

Figure 2 presents layered architecture of our intelligent agent, showing the major functionalities that will be implemented internally (left side). How these functionalities are mapped to a layered management where we strength end user participation, a middle-ware management, and the instrumentation of services and resources. On the right side of the figure we map our architecture to a hypothetical hierarchical Knowledge Base of a layered Intelligent Agent.

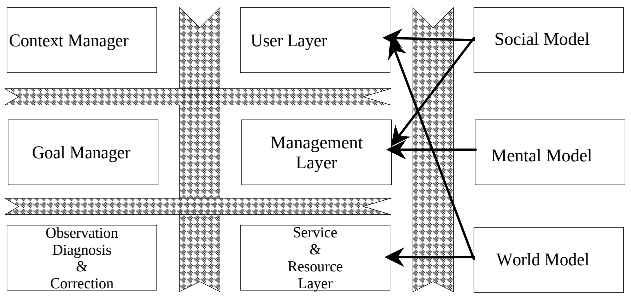

Figure 2:Intelligent Agent (management front-end) layered architecture. Three layers are showed accordingly to the different points of view: internal structure, management framework and intelligent behavior

Organizing the distribution of IAs

It is 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.

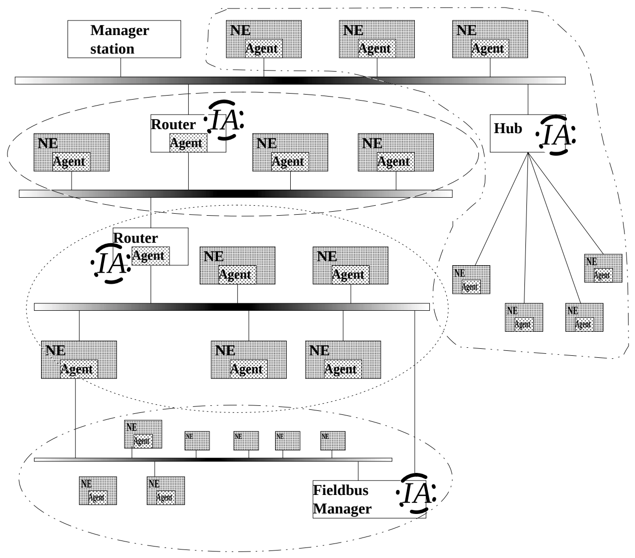

Figure 3:Intelligent Agents spread over a typical management environment.

4 Intelligent Agents

Before starting to describe what should be the role of an IA in the network environment it is worth to clarify why do we have chosen IA instead of other entity type for a management process at an intermediate level, between managers and agents. The term *Agent* for the network management community is an entity keeping managed objects as a set of structured data known as a management information base (MIB), which are an abstraction of network resources, properties and states for the purpose of management [6]. However, a more general view of the agent concept would point us to a broader definition. Hence, the agent concept enjoys the following properties [12]:

- **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 qualifying them as Intelligent.

As already claimed, NMS in general suffer from the inability of human 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 is easy to understand that the entities that are in charge to substitute human beings should have the properties mentioned above.

4.1 Application context

Before going through the role of an application context it is worth-while to make clear some terminology used hereafter.

- **Application:** A group of processes and services that perform a defined business function (manufacturing scheduling, shop-floor controller, video-on-demand, etc)
- **Service:** A generalized function that is potentially useful to a number of applications, and can be invoked from a service provider (e.g a database, NFS server, RPC,...). A service may rely on other services to help complete its task.
- **Client:** A process that uses one or more services to accomplish its task.
- **Process:** An instance of a client or single process service provider. Any given client or service provider may have multiple process instances active at any one time, whether distributed or in the same local.

User requirements are delivered to an IA (NMS) through application contexts in the same way as the ACSE [1] definition for the establishment of peer to peer associations. For the NMS an application is the smallest unit being considered to collect user requirements, no matter how many processes are needed to build it. However, it is possible that a distributed application could provide the NMS with several contexts, all associated with the same application.

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

- *Application context header*
- *Sequence of required Services*
  - *Required Service QoS*
    - ∗ *Sequence of QoS Dimensions*
      - · *Availability*
      - · *Timeliness*
      - · *Performance*
      - · ...
  - *Service fault severity*
  - *Sequence of resources offering 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 dedicated test procedures for services or resources. For example, to test network resources, like temperature sensors on a field bus (non instrumented NEs), the IA will be pointed to the appropriate testing procedure.

Context decomposition

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

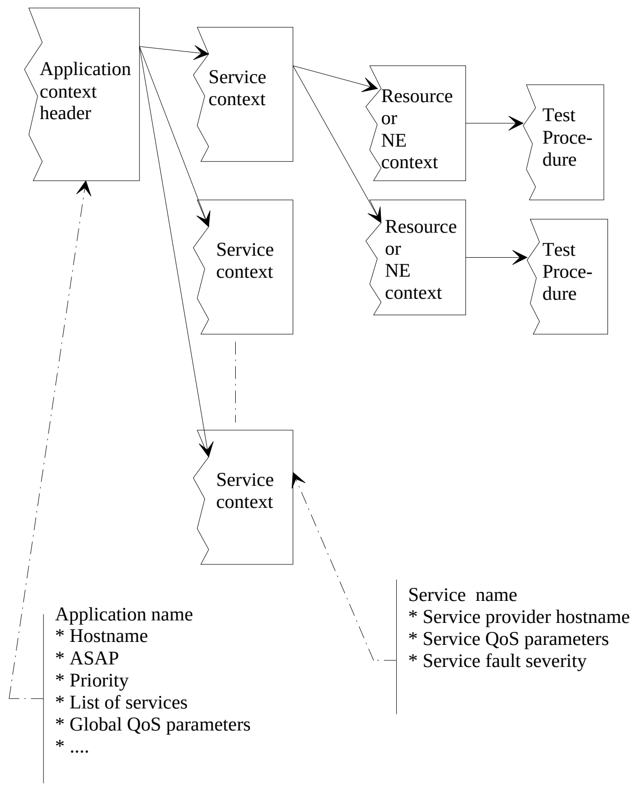

Figure 5:Application context organization in sub contexts.

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 may be used also to reduce 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.

QoS role in application contexts

So far, one may notice that quality of service is almost what can be found in applications contexts. In fact, to have an effective network and systems management, the critical issue is to have a precise knowledge of applications QoS profiles, otherwise every analyze made over monitored data on the networked environment risks to be unrealistic. Thus it is very important to introduce a view of the QoS in this framework [4].

There are several aspects or dimensions to QoS in which this framework should focus. We will present three of them as a good starting point:

- **Timeliness -** This aspect is concerned with different forms of timing constraints that may be found in applications, such as bounded response time, periodicity of events, and temporal relationships between events.

- **Availability -** This aspect is concerned with maximizing the likelihood that a service is available for use when a application attempts to use the service.

- **High performance -** This aspect is concerned with performance features such as throughput.

It is convenient to make a clear distinction between QoS issues relevant to computational or engineering viewpoints. In our network management framework, we are only concerned with the computational viewpoint, where QoS parameters (guarantees) required by applications (provided by services) are stated declaratively as service attributes.

Each dimension of QoS has associated its own set of parameters. We could also organize these parameters in QoS domains which are the computational models for a QoS dimension. A QoS domain is also specified by a set of attributes, called QoS attributes. So the organization of applications contexts must reflect this organization, although in figures 5 and 4 the presentation were more informal for the sake of comprehension.

It is also important to make the distinction between QoS offered and QoS expected (or required)[2]. So the NMS should capture them in QoS specifications from either services and applications.

4.2 Intelligent Agent goals

IA in spite of their "intelligence" must not widespread their monitoring efforts, through all data available over the network (in MIBs located on the NEs). This is the major problem of a static approach 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 is able to 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.

[2]Offering and expecting constitutes the basic difference between the QoS specified on contexts either by customer or service applications

Cooperative behavior and goals delegation

Dependently of network organization in domains some of the received application contexts could entail the cooperation with others IAs. Thus as result the IA might create goals to satisfy the contexts it is in charge of, and forward some of these goals to others and more appropriate IAs. Nevertheless, delegated goals responsibility still belong to the goal creator. For instance, if the cooperating IA is not able to guaranty the delegated goal, he must notify the goal owner, which will notify the applications concerned.

Goals principle

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 low level operations that the IA will perform to verify goal achievement. The homogeneity of the goals representation yields a separation of user requirements from distinct information models and heterogeneous management definitions, for services and 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, it 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 we can imagine that even IAs based on distinct instrumentation technologies can still cooperate.

4.3 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 [3] or SNMP [2]. Other protocols or communication mechanisms such as RPC [9] are also eligible to this communication axe. The approach sends the problem to the formalization of information structures that will be able to carry the contexts. The challenge is to design these structures in a way that they could host all possible contexts required by applications. From figure 4 and figure 5, we can say that this does not seem to be the more 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 or fix misbehaving network services or resources, as well as rules that will enable the IA to identify faults or symptoms of oncoming faults. In fact what IAs need are ways to communicate attitudes about information, such as querying, stating, believing, requiring, achieving, subscribing and offering.

These information attitudes have not any means to be carried on through protocols such as CMIP and SNMP. Intelligent Agents research community have been dealing for a 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) [10] which uses KIF (Knowledge Interchange Format) expressions appropriate to this type of communication.

KQML provides agent designers with a standard syntax for messages, and a number of performatives that define the force of a message. KQML can be used in any environment where software agents need to communicate something more than pre-defined and fixed statements of facts and provides dynamic run-time interaction, so that intelligent agents can combine their efforts, or make use of other agents abilities in order to achieve their goals[10].

We do not 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 is clear that an IA as an autonomous entity able to perform its tasks alone will lead, of course, to a complex architecture. We identify several major functional blocks for IA architecture that actually seem to be of major importance.

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

Context manager

Contexts manager (CM) is responsible for maintaining a repository 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) had not been successfully guaranteed (probably expressed with distinct syntaxes).

Goal manager

The goal 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.

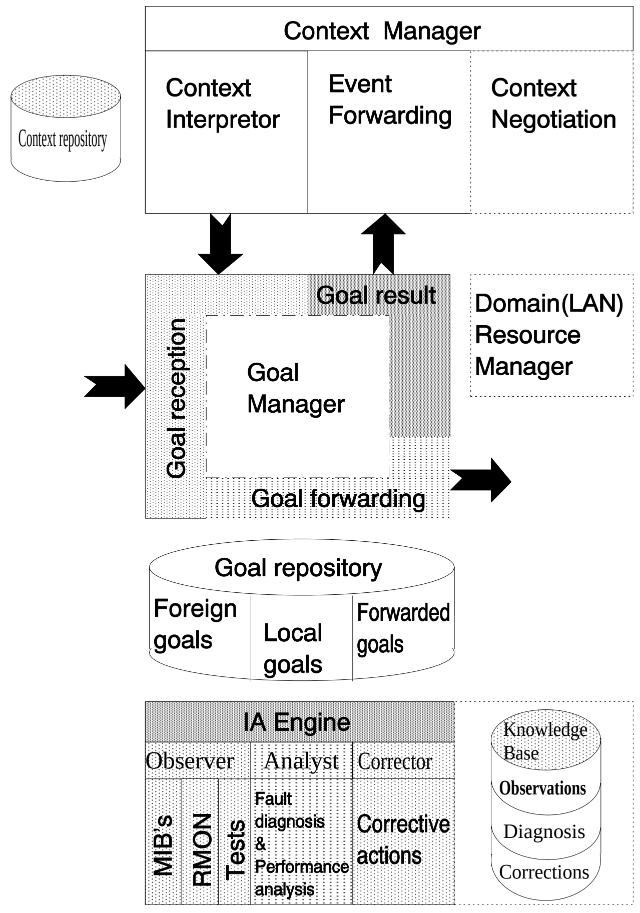

Figure 6:Intelligent Agent internal architecture.

Engine

The engine is organized in three functional blocks:

- **Observer** 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 is not 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 failure 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 generated by agents should be sent to the IA instead of to a manager station. So the analyst beyond alarms[3] generated by the observer also has as 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 is in fact too much to a human being to process manually all these alarms, but also for an automatic centralized process it is not an easy task accordingly to the multiple sources of alarms and their non evident relationships and side effects between them.

[3]Notification of unsuccessful goal achievement

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 to management operations. Since goals are independent from any particular SMI (Structure of Management Information) or MIM (Management Information Model), there might exist several mappings for the same management goal.

Like most of our framework, which is network service oriented, so are the knowledge in the IA (ontology oriented). This means that each time an IA is engaged with the management of a determined service it will try to have the knowledge about how to manage it.

IA knowledge is divided in three main areas: monitoring, diagnosing and fixing. The knowledge can be stored under different forms: management procedures (scripts, management object classes[4]) and rules. The combination of these types of knowledge is perhaps the best approach, avoiding the storage of huge quantities of rules.

Any solution that intends to build an IA (or a general management application) with all the management knowledge or expertise needed to manage the surrounding environment is not feasible to succeed. Usually leads to very heavy solutions in which it is tried to preview all the future misbehaviors, whether by storing locally all the expertise available as well as all known problems.

On the contrary we think that an IA must be a light management application with the minimum of knowledge stored locally, and fully adapted to its surrounding environment. This means that the IA must have access to remote knowledge sources. We can suppose that in a corporate network the top level managers are a possibility. However, in a networked world such our world today it might make sense to lookup for this knowledge also directly on the service provider or resource implementor.

6 Conclusion

Customizing network and systems management is not an easy task, but is in fact what is missing after several

[4]the likelihood between management object classes and scripts should be noticed

years of work done at instrumentation, structures for management information and protocol levels. The framework presented in this article tries to open new roads on network and systems management supported in two major ideas: automated management and application requirements. This article has presented a novel approach for network and systems management, in which user applications requirements for can be taken in account. These requirements are passed as application's contexts to Intelligent Agents running in scoped domain of operation. The approach entailed new axes of communication for the management architecture, whose characteristics and solutions were analyzed.

Further issues

As we stated before this framework must be extended to the manager station level, which have an important role to play in the interaction with the Intelligent Agents. Since agents will perform management autonomously we will put some care with the definition of domains. As well as in order to avoid conflicts between IAs or bound the IAs degree of freedom, management policies should be considered carefully (defined and formalized) so the IAs can use them to control its behavior. These two subjects domains and policies might be an interesting contribution to the Intelligent Agent research field from the network and distributed systems management perspective.

References

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

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

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

[4] TINA Consortium. Quality of Service Framework, Draft TINA Report, November 1994.

[5] G. Goldszmidt. Distributed systems management via elastic servers. In *Third International Symposium on Integrated Network Management*, pages 95–107, 1993.

[6] Kirk Shrewsbury J. TMN in a Nutshell, July 1994.

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

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

[9] Inc Sun Microsystems. RPC: Remote Procedure Call Protocol specification version 2, RFC 1057, 1988.

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

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

[12] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[13] Y. Yemini, G. Goldszmidt, and S. Yemini. Network management by delegation. In *Second International Symposium on Integrated Network Management*, pages 95–107, 1991.