Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Logical Intelligent Management of Active Networks

Giuseppe Di Fatta, Salvatore Gaglio,Giuseppe Lo Presti,
Giuseppe Lo Re, Ignazio Selvaggio

RT-ICAR-PA-03-03 settembre 2003

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Logical Intelligent Management of Active Networks

Giuseppe Di Fatta[1], Salvatore Gaglio[2],Giuseppe Lo Presti[2], Giuseppe Lo Re[1], Ignazio Selvaggio[2]

Rapporto Tecnico N.3:
RT-ICAR-PA-03-03

Data:
settembre 2003

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo Viale delle Scienze edificio 11 90128 Palermo
[2] Università degli Studi di Palermo Dipartimento di Ingegneria Informatica Viale delle Scienze 90128 Palermo

Logical Intelligent Management of Active Networks

Giuseppe Di Fatta
Salvatore Gaglio
Giuseppe Lo Presti
Giuseppe Lo Re
Ignazio Selvaggio

Index

1 Introduction

This work focuses on improving computer network management by the adoption of artificial intelligence techniques. A logical inference system has being devised to enable automated isolation, diagnosis, and even repair of network problems, thus enhancing the reliability, performance, and security of networks. We propose a distributed multi-agent architecture for network management, where a logical reasoner acts as an external managing entity capable of directing, coordinating, and stimulating actions in an active management architecture. The active networks technology represents the lower level layer which makes possible the deployment of code which implement teleo-reactive agents, distributed across the whole network.We adopt the Situation Calculus to define a network model and the Reactive Golog language to implement the logical reasoner. An active network management architecture is used by the reasoner to inject and execute operational tasks in the network.The integrated system collects the advantages coming from logical reasoning and network programmability, and provides a powerful system capable of performing high-level management tasks in order to deal with network fault situations.

2 Overview

The enormous growth of computer networks and the emerging technologies for network programmability make the research on network management an interesting challenge.Network management is a complex activity, which very often requires the human intervention to create action management plans, to coordinate network assets, and to face fault situations. Network management applications are traditionally centralized around a managing entity, like in the Simple Network Management Protocol (SNMP) [15] approach, and very often the managing entity is a simple interface to a human operator. Several distributed architectures for network management have been proposed to relieve the load from the central management station and to distribute control tasks by means of the Active Networks technology [1], [2], [5], [17], [16]. Active networks introduce network dynamic programming and allow an easy deployment of "ad hoc" solutions in the active nodes on behalf of contingent management tasks. Given simple management tasks, such for instance multiple failure traps, data merging, automatic backup-link activation, the management architectures based on Active Networks make easy to deploy a distributed strategy. Nevertheless, for more complex tasks, it is still required the human intervention because only experts, who know the network complexity, can understand an high level management goal and plan a sequence of intermediate steps to reach the main objective. In a traditional network management environment, if a user asks to know the causes of a network failure, a network expert, according to some strategy, can query the network devices to trace the problem back to its original causes. Planning of actions, prediction and classification of events, diagnosis or explanation of failures, etc., are typical human activities that advanced Artificial Intelligence techniques are today able to provide.In our approach we adopt the Situation Calculus to represent a logical model of the network and the Golog programming language to implement a reasoner capable of accomplishing generic high-level management tasks.The logical reasoner allows not only the representation of particular network states, but also the representation of their evolution. The evolution, namely, can be represented as a transition process from a situation to another one and it is triggered by the execution of particular actions.In this work we propose a two levels framework for network management, where the upper level is represented by a logical, centralized inference system acting as an external managing entity capable of directing, coordinating, and stimulating actions in an active management architecture. To this end, it exploits the capabilities of a distributed active management framework, which represents the lower level and makes possible the deployment of code across the whole network. This way, our management architecture exploits the potential of "doing" of the Active Networks technology conjugated with the potential of "planning", which is typical of the artificial intelligent systems [8], [3].The originality of this work relies on two different aspects. Firstly, the architecture adopts a logic programming language to implement a high-level logical reasoner capable of producing failure diagnoses, or generating investigation plans to better define the decision scenario. Secondly, it is able to plan and carry out the information acquisition by exploiting the active network framework [2] that provides simple and effective tools to capture the right information in the appropriate places of the network.The remainder of this work is structured in the following way. Section 3 introduces how the situation calculus can be adopted in our scenario to model the dynamism of network events. In section 4 we describe a Network Management Framework based on Active Networks whose functionalities perfectly answer the requirement of dynamic programmability imposed by our inferential engine. Section 5 illustrates the architecture of the distributed network fault management system and section 6 reports some experimental results. Finally, section 7 traces some conclusions.

3 Intelligent Network Management

Network Management [12], [14] involves several different functions grouped in five main areas: configuration, fault, performances, security, and accounting. Current management protocols approach these functional tasks in a rigid way, generally based on centralized monitoring, analysis, and control carried out by human operation staff.This work proposes the adoption of a two levels framework, where a decision system behaves as managing entity, and active Network Management system executes the operational tasks.The upper level automation is made possible by the adoption of a logic-programming environment, which intrinsically owns special features that easily allow the achievement of tasks such as the decision of actions, the prediction and classification of events, the diagnosis or explanation of failures, etc. Intelligent network management requires a model of the network, which is able to capture both the cause-effect relationships and their dynamic nature (time varying relationships). This way, a logical inference process can use the system model to relate events that happen in the time-space to some other events which can be seen as their root causes.We adopt the situation calculus[9] to model the network and its dynamic evolution. The situation calculus is a logic language specifically designed for representing dynamically changing world. The management system we have designed can be classified as an expert system that adopts a case-based strategy [4]. It is an expert system since it owns a complete knowledge of the working environment. Namely, among the logic predicates it is necessary to provide ontological descriptions for all the entities which populate the external world, and for all their relationships. Moreover, we provided the system with the further capability of retrieving new knowledge on the basis of the current situation where the world lies: in particular, if the information available in the knowledge base is not sufficient to reach some deductive goals, new data are required and successively acquired by means of specific sensors positioned in opportune nodes of the network. The twofold nature of the system allowed us to achieve a simple and accurate monitoring of the managed network. In particular, we exploit the noticeable capabilities offered by the Reactive Golog [13] language as the specific reasoning environment adopted to implement the system. The adoption of Reactive Golog language is due to its noticeable expressiveness and to its capability of providing simple and linear frameworks to the programmers.

3.1 Reactive Golog

Reactive Golog is a logic programming language, which merges the flow control and procedural constructs of an Algol like language with the logic constructs of Prolog. A Reactive Golog interpreter is easy to implement under a Prolog interpreter and makes available all the capabilities of the situation calculus.The most important components of a Reactive Golog program are:

- Primitive Actions;
- Fluents;
- Primitive Actions Preconditions;
- Successor State Axioms;
- Procedures;
- Rules.

The formalization of the world is performed through well formed formulas of the first order logic, while the dynamism is captured through the primitive concepts of state, primitive action and fluent.We can think the state as a snapshot of the world at a determined moment. All changes to the world can be seen as the result of some primitive actions.Relations whose truth-values may vary in different situations are called relational fluents. They are represented by means of predicate symbols which take a situation term as their last argument.Primitive actions preconditions are rules that describe when actions can be carried out given a state of the world. The preconditions are stated by fluents.The successor state axioms provide a complete description about the fluents evolution in response to primitive actions. They are needed for each predicate that may change its truth value over the time.Procedures represent the complex actions and constitute one of the most important features of the Reactive Golog. They allow to group long sequences of primitive actions and to implement recursive formulas. Like in the imperative languages, they uses formal parameters.Generally, dynamic systems are not totally isolated by the rest of the world, but they continually receive solicitations and they interact with the external world. The Reactive Golog rules allow these interactions describing how the world evolves when an external action is performed. This aspect is the so called "reactive behavior". We use a logical approach for the networks management to exploit the logical languages and artificial intelligence characteristics in terms of synthesis, classification facility, deductive reasoning, planning and correlation of various events, diagnosis in the failures management. To this end it is necessary to produce an ontology to create a network model and to capture all the cause-effect relationships and the events dynamism.

3.2 Ontology

To represent the ontology we use the frame. The frames are an artificial intelligence formalism to represent general knowledge that is modified for keeping the current situation. The frames are collections of attributes and values that

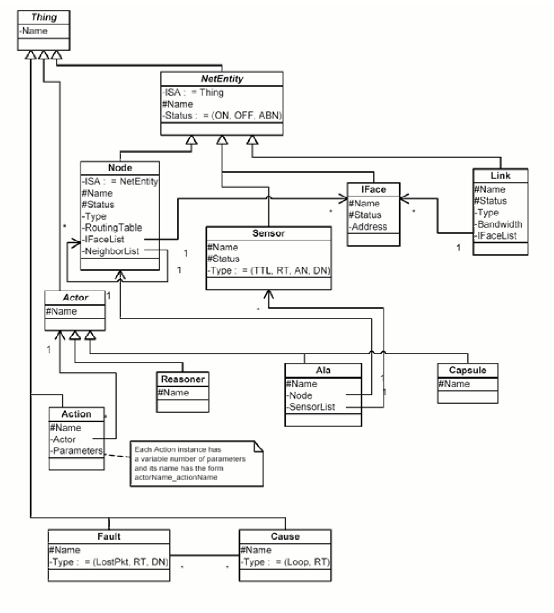

Fig. 1. *Frame*

describe some world entities.Frame fundamental characteristic is the heredity, specialization of general concepts that contain information and characteristics shared by a elements set.The frames framework is shown in the figure 1. All entities are specializations of the most general entity "Thing" that has the alone slot "name" used for identifying various frames. The first level contains the frames: Fault to describe the failures that can occured on the network; Causes for the various failure causes; Action to model the possible actions on either network and logical reasoner; the frame Actor is used for describing the various agents (Network, Capsules, Reasoner, Wing); NetEntity finally contains all the characteristics shared by various network entities (Node, Sensor,Iface,Link).

3.3 An Example of Network Modeling

One of the key challenges of our work is the construction of a logic model capable of fitting as more as possible networking concepts. To this end, we have formalized all the entities which constitute the network layer of the OSI reference model, and all their basic capabilities.The ontological engineering process about which network aspects should be represented and which form of representation could present the most suitable features, induced us to establish a relationship between the functional layers of the OSI networking model and some correspondent layers of dynamic knowledge representation.In this vision, the lower level view concerns the physical features of the network, while, for instance, routing devices and their connecting communication links represent knowledge at a higher level.Furthermore, this first amount of knowledge representation has been integrated with the capability of representing the functioning during the time. Logical sentences, whose validity is bounded to the time, have been introduced for representing the temporal status of a given node or a particular link. In order to represent the network as a dynamic system capable of flowing from a situation (current state) to another one (successor state), we imagined the network as an active entity capable of carrying out actions which modify its own configuration. According to the Situation Calculus, we selected the atomic actions that the network is able to carry out, and we described the state transitions involved by the actions execution. We defined some basic actions for managing the changes in the status of the basic network elements such as nodes, interfaces, links, sensors, active local agents (ALAs), routing tables, etc.Moreover a set of special purpose actions were defined to allow the notification of network monitored events to the logical reasoning system. These actions are performed by the network, thought as an active entity.
As previously described, the Situation Calculus allows the distinction among different temporal situations in which a single predicate may or may not be verified. This differentiation is achieved by means of the current situation as last argument of the fluents. For instance, the *Net_Link* and *Routing_table* predicates, respectively used to represent networks links and routing tables of the network routers, have to be written in the fluent form because their validity depends from the current situation. Here, in order to give an idea of the logical design process we produce a simple example. In the example we show as our logical system manages the routing tables update process for keeping traces of the changes in the networks. To this end, two primitive actions, define_rip_table(N,T) and update_rip_table(N, Upd) are shown.

Primitive Actions.

```
primitive_action(define_rip_table(N,T)).
primitive_action(update_rip_table(N,Upd)).
```

The first action is used to instantiate the fluent predicate "routing_table(N,T,s)" in the situation s resulting from its own execution. This way, the logical reasoner will achieve the knowledge that, in the situation s, T is the routing table of the node N.The second action will update the same fluent. Namely, if the fluent "routing_table(N,T,s)" is verified, the update_rip_table(N,Upd) predicate will update the old value of the node N routing tables with the Upd information.The above actions require the following preconditions for their correct usage in the situation s of the network:

Primitive Actions Preconditions

```
poss(define_rip_table(N,T),S):- not routing_table(N,_,S).
poss(update_rip_table(N,Upd),S) :- routing_table(N,_,S).
```

The preconditions are determined by the truth-value assumed by the fluent predicate in the situation s and their validity enables the action executions. In the above example, the action define_rip_table(N,T) is possible in the situation s if the fluent routing_table(N,_,S) assumes a False value in this situation. Likewise for the action update_rip_table(N,Upd), which instead will be possible when the routing_table(N,_,S) fluent assumes a True value.The State Successor Axioms are used to define the effects of the actions over the world, and their definition is necessary for each fluent.For instance the routing_table(N,T) fluent will assume a True value if the define_rip_table(N,T) action has occurred or if a previous routing_table(N,Old) has been modified by the update_rip_table(N,Upd) action.

State Successor Axioms

```
routing_table(N,T,do(A,S)) :-
        (routing_table(N,T,S), not A=update_rip_table(N,_)) ;
        ;((A = define_rip_table(N,T)) ;
          ;(A = update_rip_table(N,Upd),routing_table(N, Old,S),tab_merge(Upd, Old,T))).
```

4 Active Networks Management Framework

In this section we describe the management architecture for Active Networks which has been adopted to perform the operational tasks required by the managing entity.Active Networks introduce programmability in the network; new software components can be dynamically injected in the network nodes. In our active management framework we exploit network programmability in order to support distributed, cooperative, and adaptive management applications. The general framework is shown in figure 2. The logical reasoner, which operates as a Managing Entity (ME), sends queries and receives replies in the Extensible Markup Language (XML) to and from an AN Access Point. The AN Access Point is an active node hosting a Gateway service which performs two basic

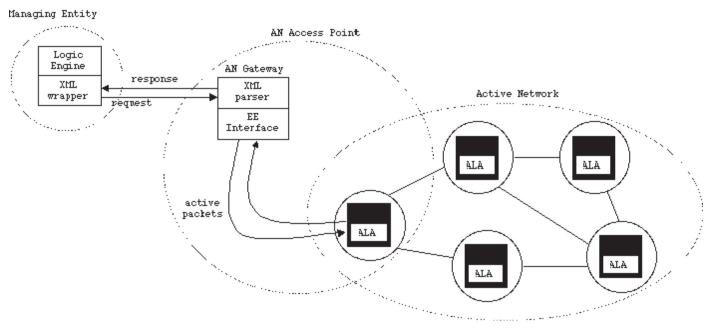

Fig. 2. *Active Network Management Framework*

tasks: the translation of XML requests to the specific language adopted by the Execution Environment (EE) and the injection in the network of the appropriate active packets to accomplish ME requests.The Gateway services provide the interface to a particular active network implementation (EE) and it is specific for the supported network programming language. In general, several nodes in an active network can be configured to provide Gateway services.As illustrated in figure 2, a resident management service, the Active Local Agent (ALA), is resident in each active node. ME and ALA are the end points of the management communication. Namely, the ME can either query the Active Local Agents (polling) or deploy subtasks to them (programmable trapping). Local agents asynchronously perform subtasks in terms of actions to be executed at local events occurrences. The Active Local Agents can be seen as real teleo-reactive programs [11] which are installed and executed in the network nodes.Namely, the internal mechanism of ALAs involves the installation of predetermined local variables which implement the conditional statement of teleo-reactive agents. These variables constitute the discriminating values over which filters are installed in order to generate events, that in turn cause the execution of some actions. Finally, actions may install further conditions which will be evaluated till the predefined goal is achieved, i.e. the monitoring of fault root causes.
The architecture can be shared by different AN implementations and provides a common management framework for different EEs. We adopted XML to define a set of requests/replies for basic operational tasks, which are common to any Active Network environment. The Gateway basic services currently available are described in [2]. However, the set of Gateway services can be gradually enlarged. Once a new service is developed and tested, it can be provided for public use. Each basic service requires an EE-specific code fragment stored at the Gateway. This way, Gateway nodes provide transparent access to different Active Networks. For instance, the ME can use Gateway services to discover the network topology, explore the network nodes, find out which active applications are running and monitor their activities. We are particularly interested in managing those services, which are able to carry out specific tasks on behalf

of the managing entity, such as the retrieval of a particular information from a node, or the verification of compound tests on several routers. For instance, with reference to the example of section 3.3, we implemented an active service, which allows the tracing of the state of a link in a given temporal interval $[t_1, t_2]$.The system dynamically deploys its components in the active nodes to implement a distributed strategy for a given goal. Two aspects are worth to be noticed: the reasoner adopts the active management architecture to exploit the programmability of network nodes and deploy ad-hoc tasks for a given goal; on the other hand the active management infrastructure can be used to manage, to monitor and to debug distributed network applications.A key point of this architecture is that the logic reasoner is completely decoupled from the sensors and agents implementation, allowing it easy to deploy the reasoner in other network environments. Namely, the flexibility offered by logic languages allows the logical reasoner to interact with other environments, provided that a small interface has to be implemented to gather data and send commands to that environment. In the current implementation the interface role is carried out by a Gateway service [rif. AIxIA/AMS] which is available on one or more network nodes, and it is able to translate XML-based queries to AN environment specific capsules and vice versa.

5 Network Faults Management System

The aim of our system, is related to the knowledge management of network events. To this end it deals with large archives of events obtained from the network systems, and extracts useful information from these data.Goals of such a system are:

- to diagnose root causes of network faults and performance degradations by establishing relationships between network events;
- to filter event (alarm) flood by correlating several events into a single conceptual event;
- to retrieve further ‘ad hoc’ diagnostic information;
- to adopt repair actions.

Such a system should provide:

- correctness: the root causes inferred by the system should be entailed by the detected events with a high likelihood, i.e. the root causes have really occurred in the network;
- optimality: the system should infer as small a set of root causes that can explain all the detected events.

The knowledge of network events constitutes the necessary information to answer questions about the real causes of network failures.Faults management is based on the definition of the normal operating conditions, and on the abnormalities detection. In general, alarms are generated in the network when abnormalities are detected. In alarm-based fault detection systems, a single fault will often cause a large number of alarms. Moreover, several faults may coexist causing a cascade of alarms. Our system is able to correlate alarms, to pinpoint their root causes in order to efficiently handle them. The logical inference process starts with a initial phase of information retrieval allowing the system to acquire the basic data over which can be established the successive reasoning processes. In analogy with an human behavior this step may correspond to the initial observation performed by an human operator to achieve an overall knowledge of the reasoning domain. The macroscopic behavior of the logical system is shown in the figure 3. In the following, the main steps involved in the monitoring of faults will be described. The process of faults management can be distinguished in a

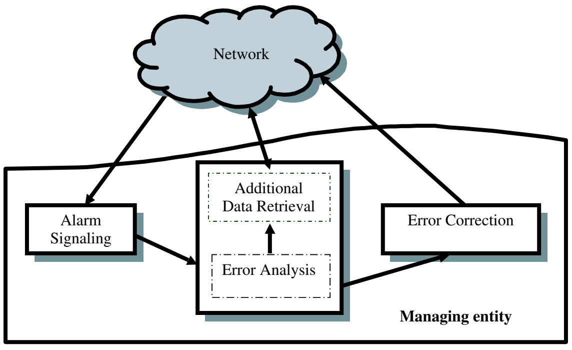

Fig. 3. *The logical inference process*

succession of steps [6] related to the alarm condition signaling, the error analysis with its consequential step of acquisition of additional data useful for a correct diagnosis, and finally the correct fault recognition and correction.

Alarm Signaling In this phase the signal of a network failure can be risen either by an user or by a sensor previously installed, which for instance may signal to the Managing Entity that a packet has been discarded on a router of the network.

Error Analysis Once the failure has been notified, the Managing Entity using the previously stored information tries an inferential process in order to determine the root cause of the failure. In the positive case, the fault is recognized and the successive step is performed. In the negative case, i.e. the knowledge base does not contain the right elements to deduce the root cause, the Managing Entity produces actions devoted to the positioning of new sensors with the aim to collect new diagnostic elements. The whole process is repeated until the Managing Entity is able to determine the failure nature.

Error Correction After the Error Analysis has produced the diagnostic inference, some predefined actions are taken in order to fix the discovered failure. In the current version, the system is capable of managing the following types of events:

- Errors in the RIP routing tables of the network routers;
- Early packet discard for expired TTL;
- Full node failure;
- Full link failure;
- Loops in the RIP routing tables of the network routers;
- Changes of the state of neighbor routers.

To demonstrate the effectiveness of the automated reasoning system we analyze in more detail its behavior in a sample case of functioning. The capabilities of discovering the root causes of failure and of gathering additional data required by the reasoning process can be illustrated by the following example. Let us suppose that a routing loop has been generated on the network. The intervention of the Managing Entity can be triggered by a symptom, in this case packets discarding for expired TTL. The $primitive\_action(lost\_pkt\_up(N))$ will notify this occurrence. Reasoning on the imagine routing tables maintained in the knowledge base, the ME is able to discover the loop occurrence. Namely, when a packet is discarded, the sensor installed on the involved router notifies the abnormal condition to the logical reasoner sending to it source and destination of the packet. Using this information the reasoner can simulate a journey trough the path from the source to the destination, using the routing tables of the encountered nodes. Throughout the simulation it marks each node traversed along the path, and if a previously marked node is met, the reasoner infers that a loop exits in the network. Furthermore, if throughout the simulation some routing table is missing, the reasoner may require these data to the opportune nodes by means of appropriate exogenous Golog actions, thus exploiting the Situation Calculus paradigm. Furthermore, in order to detect the node which is responsible of the failure, the actions taken by Managing Entity will consist of the routing tables changes retrieval by means of the following preocedure:

$$proc(collect(List), ?([] = List)\#?([X|Tail] = List) : get\_local\_var(X) : collect(Tail)).$$

In the above procedure List contains all the nodes which belong to the loop. In this procedure, firstly, it is considered the case of empty list. Otherwise, List is assumed as $[X|Tail]$, the external predicate $get\_local\_var(X)$ is invoked in order to obtain further information about the node X routing table history, and finally the recursive call $collect(Tail)$ is executed. The collected changes are stored in the fluent $repository(\_,\_)$ which has the following state successor axiom.

$$\begin{aligned} repository(L, Do(A,S))\text{:} &-repository(L,S), not A = update\_repository(\_,\_,\_);\\ &; A = update\_repository(N, [Dest, Neigh, Cost], Time),\\ &, repository(Old, S), L\ is[[N, [Dest, Neigh, Cost], Time]|Old]. \end{aligned}$$

This way, the system, by reasoning on the wider scenario enriched by the additional data, will be capable of singling out the root cause of the failure using the following two procedures:

$proc(select(List, Rep), ?([\,] = Rep)\#?([[N, [Dest, Neigh, Cost], Time]|Tail] = Rep)$:
$: (?(member(Neigh, List)) : add\_causes(N, Time)$:
$: select(List, Tail)\#select(List, Tail))).$

and

$proc(check\_cause(List), ?(repository(R))$: $select(List, R)$:$?(causes(C))$:
$: select\_old(C, [N, T])).$

The former is responsible of extracting from the repository all the changes that may have provoked the loop. The latter is capable of selecting the root initial cause which has produced the routing tables anomaly.
Finally, the $proc(set\_table(N, [Dest, Neigh, Cost]))$ will be the action adopted by the Managing Entity for the correction of the routing entry.

6 Experimental Results

In order to test and evaluate our logical inference system we used an experimental cluster of 40 active network nodes which allows the setup of "ad hoc" topologies. In particular, we set up an experimental test-bed constituted by an active network with 20 nodes (routers and end-hosts) managed by the "pland" [7] execution environment and super-visioned by the reasoner by means of the ANGate [2] software package. We performed a series of experimental tests devoted to determine the reliability degree of our system, in terms of discovered faults and performed repair actions. Fault events are generated according to a Poisson distribution for their temporal occurrences, whereas we use a uniform distribution for their spatial positioning. The failures generated are those described in the previous section and for each of them a series of 25 trials has been executed.

Summary of failure detection

Table 6 shows the results of the experiments. For each managed kind of failure we report the percentage of cases discovered and the average time elapsed before the failure recognition. Most of the events are captured the full percentage of the cases. In the cases of events directly discovered by the Active Local Agents on the nodes, for instance the case of changes in the states of neighbor routers, the discovering times are constant since they depend on the sampling time of the monitoring activity. The low percentage for the case of full node failure is to be attributed to the simultaneous failures of several nodes. Namely, the occurrence of such events may provoke a disconnection in the network, thus denying the capability of retrieving the necessary alarms. Last row shows the measures of the action adopted to recover a link failure. The average time of 35 *sec.* is the overall time since the full link failure, thus meaning that the recovery time is reduced to 5 *sec.*. Finally, the overall percentage of false positives, also known as false detections or false alarms, has been measured and it is limited to the 1% of all the experimental cases.

7 Conclusions

The novelty of our project arises from the original idea of complementing a logical reasoner with the versatility of active networks. The integrated system collects the advantages coming from logical reasoning and network programmability, and realizes a powerful system capable of performing high-level management tasks and dealing with unusual network situations better than traditional management systems.The logical reasoner is, namely, able to deduce knowledge and find correlations from data and events which are distributed on different places of the networks and which occurred in different instants. The inferential engine provides reasoning on a high-level network model and behaves as an intelligent management agent, which coordinates the management activities at the low-level active network infrastructure. The framework allows the programming of intelligent management entities, which adopt the active networks management framework for the sensorial and actuator tasks, and the inferential logical system for the high-level behavior reasoner.

References

1. Al Shaer E., "Active Management Framework for Distributed Multimedia Systems", Journal of Networks and Systems Management, vol. 8 n. 1, 2000, pp.49-72.
2. A. Barone, P. Chirco, G. Di Fatta, G. Lo Re, A Management Architecture For Active Networks, Proc. of IEEE Workshop on Active Middleware Services. Edinburgh, UK, July 2002.
3. Covo, A. A, Moruzzi, T. M., Peterson, E. D., "AI-assisted Telecommunications Network Management", In IEEE Global Telecommunications Conference (GLOBECOM 89), pages 487–496, Dallas, TX, USA, Nov 1989.
4. Cebulka, K.D., Muller, M.J., and Riley, C.A. (1989). Applications of artificial intelligence for meeting network management challenges of the 1990s. In IEEE Global Telecommunications Conference (GLOBECOM 89), pages 501–506, Dallas, TX, USA, Nov 1989.
5. Kawamura R., Stadler R., "Active Distributed Management for IP Networks", IEEE Communications Magazine, vol. 38, N. 4. April 2000, pp. 114-121.
6. L. Kerschberg, R. Baum, A. Waisanen, I. Huang and J. Yoon, "Managing Faults in Telecommunications Networks: A Taxonomy to Knowledge-Based Approaches", IEEE, pp. 779-784, 1991
7. Hick, M., et al, "PLAN: A Packet Language for Active Networks", Proc. of 3rd ACM SIGPLAN International Conference on Functional Programming, pages 86-93. ACM, September 1998.
8. Mazumdar, S., Lazar, A., "Objective-Driven Monitoring for Broadband Networks", IEEE Transactions on Knowledge and Data Engineering, vol. 8, n. 3 june 1996, pp. 391 - 402
9. J. McCarthy, "Situations, actions and causal laws", Technical Report Stanford University 1963. Reprinted in Semantic Information Processing, (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417
10. Mills D. L., "On the accuracy of Clocks Synchronized by the Network Time Protocol in the Internet System", ACM Computer Communication Review, vol. 20, no. 1, pp. 65-75, Jan. 1990.

11. Nilson N. J., "Intelligenza Artificiale", Apogeo, 2002
12. Raman Lakshmi, OSI Systems and Network Management, IEEE Communications Magazine, March 1998 Vol.36, N.3, pp.46-53.
13. Reiter Raymond, "Knowledge in action: Logical Foundations for specifying and implementing Dynamical Systems" The MIT Press Cambridge Massachusetts 2001
14. Sidor, D. J., TMN standards: Satisfying Todays Need While Preparing for Tomorrow, IEEE Communications Magazine, March 1998 Vol.36, N.3, pp. 54-64.
15. Stallings W., "SNMP and SNMPv2: The Infrastructure for Network Management", IEEE Communicatiions Magazine, vol. 36, N. 3 March 1998, pp. 37-45.
16. Tennenhouse, D. L., Smith, J.M., Sincoskie, W.D., Wetherall D.J., Minde, G.J.: "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, January 1997, 8
17. Tennenhouse, D. L., Wetherall, D.J.: "Towards an Active Network Architecture", Computer Communication Review, Vol. 26, No. 2, April 1996