Journal of Network and Systems Management, Vol. 14, No. 1, March 2006 (© 2006)
DOI: 10.1007/s10922-005-9012-7

A Logical Architecture for Active Network Management

Salvatore Gaglio,[1,2] Luca Gatani,[1] Giuseppe Lo Re,[1,2,3] and Alfonso Urso[2]

Published online: 4 April 2006

This paper focuses on improving network management by exploiting the potential of "doing" of the Active Networks technology, together with the potential of "planning," which is typical of the artificial intelligent systems. We propose a distributed multiagent architecture for Active Network management, which exploits the dynamic reasoning capabilities of the Situation Calculus in order to emulate the reactive behavior of a human expert to fault situations. The information related to network events is generated by programmable sensors deployed across the network. A logical entity collects this information, in order to merge it with general domain knowledge, with a view to identifying the root causes of faults, and to deciding on reparative actions. The logical inference system has been devised to carry out automated isolation, diagnosis, and even repair of network anomalies, thus enhancing the reliability, performance, and security of the network. Experimental results illustrate the Reasoner capability of correctly recognizing fault situations and undertaking management actions.

KEY WORDS: Programmable networks; Intelligent systems; Situation calculus; Network ontology.

1. INTRODUCTION

Network management is not only an increasingly important, but also difficult and demanding task on modern network infrastructures. It is a complex activity, which very often requires the human intervention to create management plans, to coordinate network assets, and to face up to fault situations. Because of the increasing cost of network downtime and the complexity of deployed systems, it has become crucial to find a reliable way of managing communication networks and their services. These systems need constant monitoring and probing for the

[1] Dip. di Ingegneria Informatica, Università di Palermo, Italy.
[2] Istituto di Calcolo e Reti ad Alte Prestazioni, C.N.R. Viale delle Scienze, 90128, Palermo, Italy.
[3] To whom correspondence should be addressed at Dip. di Ingegneria Informatica, Università di Palermo, Italy; E-mail: lore@unipa.it.

purposes of management, particularly for configuration setting, fault diagnosis, and performance evaluation, but, as the size of networks increases, it becomes more and more difficult to extract the right information from them. Conventional network management facilities have been provided by network vendors in ad hoc ways, which rely heavily on human effort. However, as networked installations become larger, more complex, and more heterogeneous, manual network management is no longer able to cope and conventional approaches are therefore no longer effective. The complexity of such systems raises the cost of network management and requires the use of automated standardized tools that can be used in complex scenarios, across a broad variety of product types. The main purposes of network management are to maintain a network in healthy operational condition, to monitor the network status, and to control the network so as to maximize its efficiency. Current network management systems are typically designed according to a centralized paradigm, where a central station (manager) collects, aggregates and processes data retrieved from physically distributed devices (agents). Widely deployed standards, such as the Simple Network Management Protocol (SNMP) [1] of the TCP/IP protocol suite, or the Common Management Information Protocol (CMIP) [2] of the OSI reference model, are designed according to this strict centralized model. However, the centralized approach is characterized by a low degree of flexibility and re-configurability, suffering from severe inefficiencies and scalability limitations: the process of data collection and analysis typically involves massive transfers of data causing considerable strain on network throughput, as well as processing bottlenecks at the central entity. Taken together, these problems suggest that the distribution of management intelligence would offer a rational approach to overcoming the limitations of the centralized approach. The Internet Engineering Task Force (IETF) has therefore proposed an approach, known as RMON (remote monitoring) [1], which introduces a degree of decentralization. A key aspect is that the collection of management information should be supported in a timely way, so that the system can react to performance problems. In addition, monitoring traffic should ideally have a minimal impact on the managed network. 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. Active Networks [3, 4] 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), 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 a high-level management goal and can plan a sequence of intermediate steps, in order to reach the 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. This paper proposes a distributed multiagent architecture for network management, where a logical reasoner acts as an external managing entity capable of emulating the reactive behavior of a human expert to fault situations. We adopt the well-known theory of Situation Calculus [5] to define a logical model of the network, and the Reactive Golog language [6] to implement a reasoner capable of accomplishing generic high-level management tasks. The logical reasoner allows the representation both of particular network states, and of their evolution, triggered by the execution of given actions. Our logical framework for network management is constituted by two levels, where the upper level is represented by the logical 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 the 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," typical of the artificial intelligent systems [7]. 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 both of producing failure diagnoses, and of 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 [8] that provides simple and effective tools to capture the right information in the appropriate places of the network. In the past few years, several notable projects have been proposed in the area of Active Network management: NESTOR [9], SENCOMM [10], ABLE [11], ANCORS [12], AVNMP [13, 14], and HiFi [15]. Furthermore, in the area of logical approach to network management, some proposals have been recently presented about the adoption of a higher-level knowledge representation (see, for example, [16] and [17]). The remainder of the paper is structured as follows. Section 2 introduces the general architecture proposed for network management. Section 3 presents the logical approach and the ontology model. Section 4 describes some experimental scenarios and reports main results. Finally, we draw conclusions in Section 5.

2. THE ACTIVE ARCHITECTURE FOR NETWORK MANAGEMENT

This section presents the overall architecture, which has been adopted to implement the system proposed for network monitoring and management. The general framework, depicted in Fig. 1, shows the main components of our architecture: the Logical Reasoner, the Programmable Local Agents, the Global

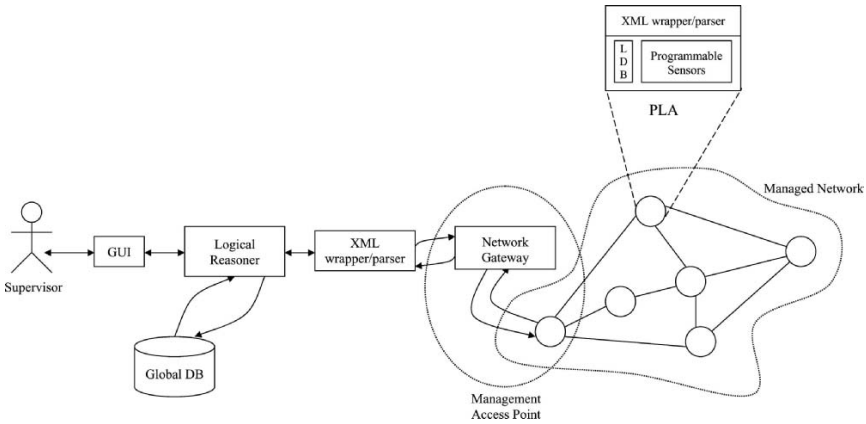

Fig. 1. The active network management architecture.

Database, and the Network Gateway. The external Logical Reasoner acts primarily as a managing entity for the system, collecting real-time data about the state of the network, and commanding further monitoring actions, in order to infer root causes and to decide suitable countermeasures. An external database is connected to the Logical Reasoner to maintain data summarization of past events. Since this Global Database (GDB) should contain only the minimal amount of meaningful information, the Logical Reasoner is equipped with an on-line filtering capability that executes data mining procedures among all the network-logged data. Furthermore, if a network user submits a query about specific anomalies, which have already occurred in the past, a new, "ad hoc" instance of the Reasoner (called Off-Line Reasoner) is executed in order to perform off-line reasoning, thus exploiting the locally logged node history and the information provided by the user. The Off-Line Reasoner is modularly designed so that it can load appropriate modules capable of managing specific queries. The results of its inference process are also stored in the GDB, thus enriching the knowledge base of network events. In order to perform the operational tasks required by the logical entity, an Active Network environment has been adopted. Active Network programmability is exploited in order to provide distributed and adaptive agents, which represent the end points of management communication. In the implementation described here, both Programmable Local Agents (PLAs), which are able to monitor each node of the network, and active codes, which perform the actions planned by the Logical Reasoner across the whole network, have been developed. Furthermore, each local agent stores the occurrences of events with their related data on its local memory. These data can be provided when a Logical Reasoner request is performed to collect data related to past behavior. The interactions between the

Logical Reasoner and the Active Network are managed by the Network Gateway service.

2.1. Logical Reasoner

The various challenges posed by network management, including real-time monitoring of network events, past event management, and planning of future activities, are considered in this section. The management activity is carried out by the Logical Reasoner that consists of two functional blocks: the On-Line Reasoner (OnLR), devoted to on-line monitoring, and the Off-Line Reasoner (OffLR), for off-line reasoning.

The OnLR is responsible for reactive behavior, and needs dynamic representation of network states that is achieved using the Situation Calculus formalism. OnLR exploits the sensors located in the network nodes to keep its network representation up to date and to collect the events which have already occurred. Using this information, it can focus its attention on specific network areas and management issues, in order to determine the causes of the observed abnormal behaviors. On the other hand, the OffLR has the main purpose of performing "a posteriori" analyses of network functioning, using the information distributed at different network nodes and already processed by different system elements. To this end, it is capable both of reconstructing the entire network state for a given temporal interval, and of examining the event flow in a limited network area. Moreover, the OffLR can perform a global reasoning process to analyze general network behavior and performance. The information deduced by means of this higher-level analysis can be used, for instance, to detect performance degradation in the communication infrastructure and to execute opportune optimizations. The OnLR main module (core module) is a lightweight reactive module which acts as a network events "sentinel," receiving notification about significant network events. In reply to such events, it performs basic reasoning activities, collecting further information about network state and, if necessary, delegating specialized logical modules to perform deeper analyses. In order to carry out this task, the OnLR can send active capsules coding command actions for PLAs. The OnLR core module starts its reasoning process whenever it detects an anomaly on the network. Anomalies reported to OnLR represent fault conditions and are associated to their local causes by means of the PLAs' distributed monitoring capabilities. OnLR can thus detect the root causes of the observed symptoms, identifying network faults that, due to their global nature, cannot be signaled by local sensors. For instance, packet losses can be interpreted as possible symptoms of network problems, such as network disconnection, routing table corruption, network congestion, or loop existence. However, it is important to notice that the OnLR core module only manages current events. It is not deputed to reason on past situations, since this more complex task is performed by OffLR.

2.2. Programmable Local Agents

In [8], it is presented a framework for network management that exploits active local and mobile agents. The monitored devices are equipped with Programmable Local Agents. Each PLA maintains a set of sensors, which can be used to monitor specific aspects of the node. Sensors for capturing early discarding of packets, discovering changes in the routing table, and detecting the state of neighboring nodes, etc., were extensively developed. It is worth noticing that exploiting the Active Networks programming capabilities, new external sensors (i.e., off-the-shelf pieces of software capable of observing certain network variables) can be plugged into the extensible PLA structure, as they become available. Internally, each local agent is modeled as a teleoreactive agent [18]. The internal mechanism adopts predefined local variables to implement the conditional statements of teleoreactive agents. These variables represent the discriminating values over which filters are installed in order to generate events, which in turn determine the execution of opportune actions [8]. Sensors are used by the OnLR to acquire information about network evolution. They perform a twofold task, registering meaningful data on local databases and notifying, when necessary, those data to the OnLR. When the OnLR starts, it always enables a basic sensor functioning level. This functionality allows to notify the occurrence of relevant events which can be considered key symptoms of fault conditions. On the other hand, when the OnLR needs some specific knowledge about the network's dynamic behavior, it "switches-on" sensors, thus enabling the notification service. The Logical Reasoner can thereby activate more specific PLA services, to perform a deeper analysis and to detect the root causes of the observed behavior.

2.3. Network Event Base

A Global Database is employed with a view to storing relevant results inferred by Logical Reasoner modules, in order to enable successive analyses of abnormal behaviors. The queries submitted to the GDB can be intended either to answer questions directly formulated by the system administrator, or to compute statistic analysis about network traffic. The OnLR can only add information to the GDB, whilst the OffLR exploits this information for its own reasoning, and it can insert new knowledge inferred by its deductive processes. The following are among the most important GDB relations: *Inference, Loop, Disconnection, RTCorruption, CongestionArea, AreaNodes, CongestedLink, FlowCongestion,* and *TrafficMatrix*. The *Inference* table maintains the inference results obtained by the OnLR. *Loop, Disconnetion,* and *RTCorruption* tables are populated both with the summarized information obtained from the data in the *Inference* table, and with the specialized inferences deduced "a posteriori" by the OffLR. The *CongestionArea* table maintains the values which represent the congestion measurements for the

monitored areas. *AreaNodes* and *CongestedLink* tables define the monitored areas as sets of nodes and connecting links. In the *FlowCongestion* table, the Reasoner reports references to those flows which are most involved in a congestion situation. The *TrafficMatrix* table contains the volume of traffic that flows between all possible source-destination pairs of the network.

Moreover, each local agent stores the occurrences of events, with their related data, on its local memory, where a Local Database (LDB) is maintained, in order to make them available whenever a Logical Reasoner request is performed to analyze past situations. All data are registered using the XML format, in order to characterize their information scope easily and to allow a fast retrieval process. For instance, in order to detect faults which have occurred in the past, the Logical Reasoner can require some additional information about the network's dynamic condition in all nodes belonging to a given area during a given interval of time. PLAs located at the interested nodes can then process the local databases and single out the required information, using XML annotation to perform data filtering.

2.4. Gateway

In order to perform its management tasks, the Logical Reasoner sends and receives XML queries and replies through a Gateway service which carries out two basic tasks: message multiplexing/demultiplexing, and reliable transmission toward the monitored devices. The Gateway service provides also the interface to different Active Network implementation, providing a common management framework for different Execution Environments (EEs). The service translates the XML requests to the specific language adopted by the Execution Environment, and it injects into the network the appropriate active packets to accomplish Reasoner requests. In order to assure fault-tolerance, several nodes in the monitored Active Network can serve as Gateway points. The set of the Gateway services can be incrementally enriched. Each basic service requires that an EE-specific code fragment is stored at the Gateway points. This way, Gateway nodes provide transparent access to different Active Network environments. The Logical Reasoner exploits the Gateway to manage distributed and programmable services, which are capable of carrying out specific tasks on its behalf (such as the retrieval of a particular item of information from a node, or the verification of compound tests on several routers).

3. A LOGICAL APPROACH FOR NETWORK MANAGEMENT

3.1. An Ontology for Networking

The philosophical notion of "Ontology" is inherited by Knowledge Engineering with the aim to describe explicit specifications of conceptualizations, where

each of them is represented by a set of definitions of elements in a domain [19]. Past efforts [7] of adopting advanced approaches to network management have revealed an absolute demand of standard representation of the domain knowledge, i.e. networking entities, protocols, their relationships, actions, events, errors, etc. In order to perform thorough reasoning on network events, our logical system needs a systematic representation of networking concepts, including hardware and software entities, and, more generally, the relationships between real- and abstract-data types. Ontological representations constitute the key for solving these challenges, constituting a framework capable of modeling concepts and relationships on some expertise domain, and providing the structural and semantic ground for computer-based processing of domain knowledge. The definition of a logical model capable of describing networking concepts is, therefore, one of the main goals of this work. The internal mechanisms of the reasoning engine must capture and reflect this ontological view of the domain. In past decades, the complex nature of networking has required the standardization of all its correlated entities and activities, and has imposed the adoption of a rigid classification in the well known Open Systems Interconnection (OSI) Reference Model, which provides a conceptual framework for computer communication. In our project we carry out the formalization of all the entities and concepts of the *data-link, network,* and *transport* layers, and of their basic features. The knowledge engineering process has led us to define the network aspects to be represented and the most suitable representation form. More precisely, we establish a relationship between the three managed functional layers 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 communication links represent knowledge at a higher level; traffic concepts at transport layer form the highest level of knowledge we considered. This basic knowledge representation is integrated with the capability of capturing the dynamic functioning. Therefore, logical sentences, whose validity depends on the time, are introduced in order to represent the temporal status of network components; moreover, in order to model the network as a dynamic system capable of flowing from a given situation (current state) to a successive one (successor state), we describe the network as an entity that can perform actions modifying its own configuration.

The expressive Web Ontology Language (OWL) [20], defined by the World Wide Web Consortium (W3C), is adopted to describe in a standard and interoperable format the ontological elements. This language results extremely suitable for the representation of general knowledge, although in our case it may depend from temporal situations. The formalized network ontology presents a hierarchical structure, that clues together classes representing network entities and associations between them. The hierarchy is shown in Fig. 2 where, for the sake of brevity, only the two upper levels are presented. Classes model *hardware* and *software* entities forming the communications infrastructure, events raised in the network,

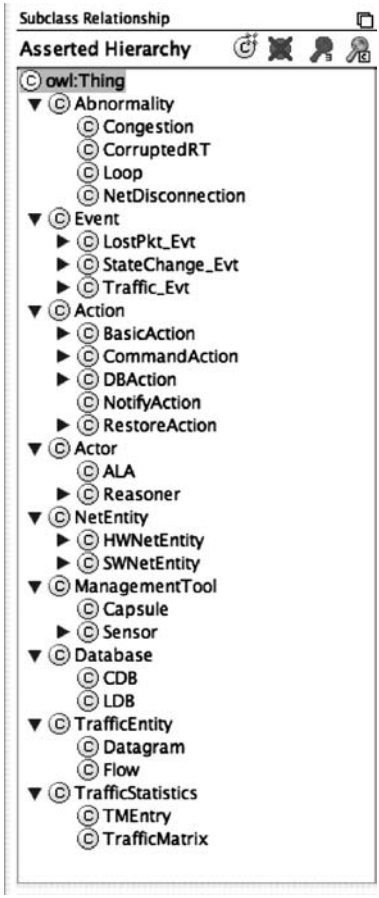

Fig. 2. Network ontology hierarchy.

actors playing into the network (managing entities and managed devices), their actions, monitoring and management tools, and traffic flowing through the network. *Hardware* entities are strictly related to the structure of the network, since they represent, for instance, network nodes (such as routers, switches, hosts, etc.), communications links, and data-link interfaces. Furthermore, relationships among classes are formalized using associations that represent categories such as "*belongs* to" (e.g. "an interface belongs to a node") or *connects* to" (e.g. "an interface is connected to another one, by means of a link"), thus defining the network topology. *Software* entities model some important, nonphysical elements, such as routing tables, queues associated with network interfaces, variables characterizing

the protocols status, datagrams, and so forth. The dynamic behavior of the network entities is represented using both class properties that model the entity status, and cross-references to the actors involved in the status modification. *Hardware* and *software* entities, expressed in the ontology, try to capture both the static, and the dynamic behavior of the network.

However, in order to understand the mechanisms adopted by the logical engine to perform reasoning about the network condition, it is necessary to model also the network actors, the management tools, and how the formers use the latters to perform management tasks. Network dynamic behavior is represented by events, occurring during the time, that are related both to entity status variations, and to network traffic.

The management architecture involves sensors deployed on the network in order to capture network events. Sensors are modeled by a subclass of *ManagementTool* class, which in turn is specialized in more subclasses related to the specific events that must be monitored. It is worth noticing that the ontology hierarchy also describes the network programmability issued by the AN paradigm through the *Capsule* subclass of the *ManagementTool* class. The class *Command Action,* subclass of the *Action* class, is defined to model the actions performed by the network managing entities. This class includes all the actions that the programmable agents in the network can perform. The *Notify Action* class, also subclass of the *Action* class, includes the actions performed by the remote managing agents in order to notify the events captured by programmed sensors to the central managing entity. Generic network events are modeled using a general class, *Event,* and they own a property describing their abnormal or normal nature. In Fig. 3, we present a schema showing the ontology classes modeling the network events managed by the system. This basic knowledge will enable the Logical Reasoner to interpret and relate several events in order to deduce their root causes, whose captured events represent the external symptoms. Among the events represented in the ontology, there are also those associated to the traffic flowing through the network. Their representation is necessary, for instance, to perform reasoning about traffic loads in the network, or to diagnose incipient congestion situations. Network traffic and related resources are modeled by the *TrafficEntity* class, that is further specialized in two subclasses, *Datagram* and *Flow.* Finally, the network data transmission involves the usage of some network resources (for instance, link bandwidth and queue buffer space). The binding between data and resources is modeled using associations between the corresponding *TrafficEntity* subclasses and *Link* and *Queue* classes.

3.2. Dynamic Logical Reasoning

The management system proposed is based on the definition of the normal operating conditions, and on the abnormalities detection. In general, alarms are

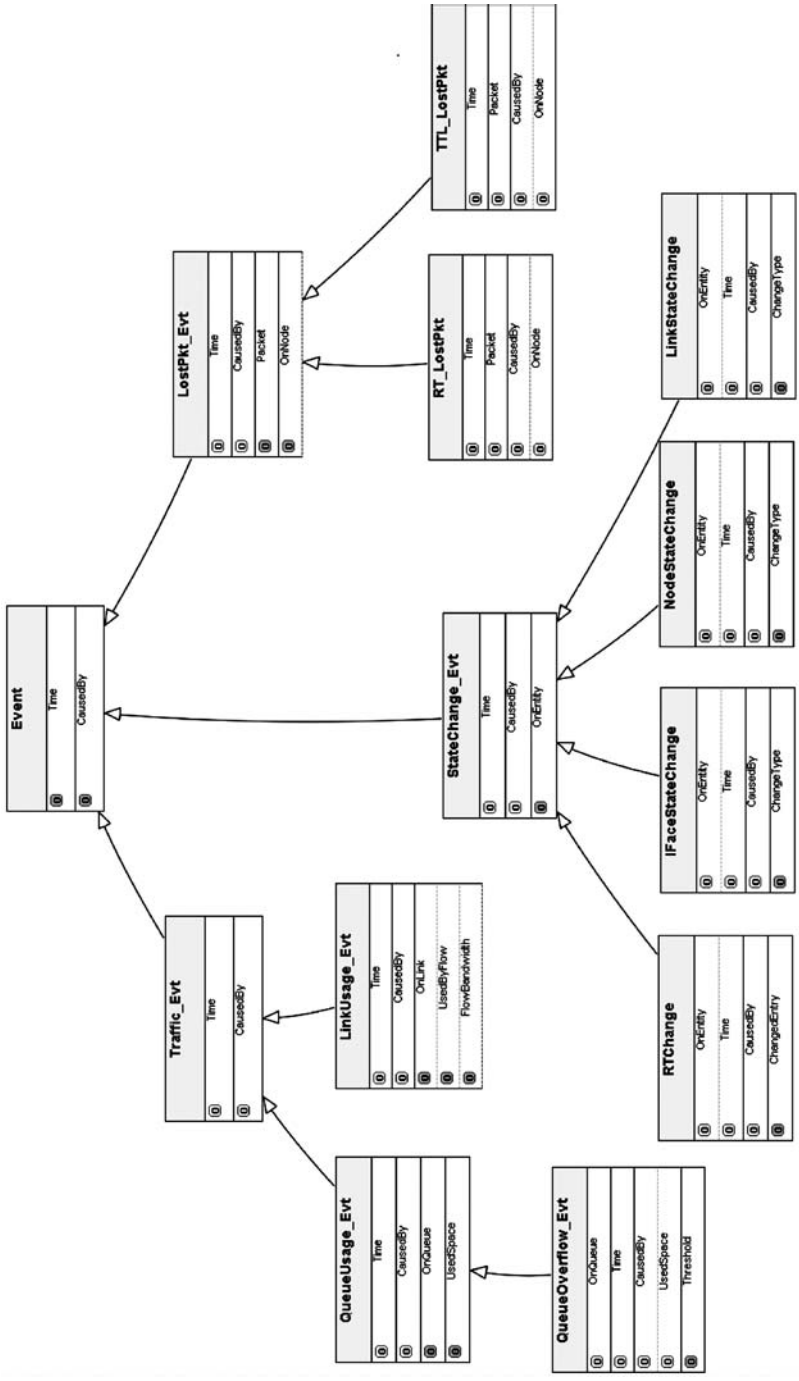

Fig. 3. Taxonomy of managed events.

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. For this reason, our system was designed with the aim to correlate alarms, and to isolate their root causes in order to efficiently handle them. The overall reasoning process starts with an initial phase of information retrieval. During this phase, the system can acquire the basic data over which it can establish successive reasoning processes. In analogy with human behavior, this phase may correspond to the initial observation performed by a human operator to achieve an overall knowledge of the reasoning domain. The process adopted to manage a network fault can be distinguished in a succession of steps [21], 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. In the *alarm signaling* 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 Logical Reasoner that a packet has been discarded on a network device. Once the failure has been notified, the Logical Reasoner, using the previously stored information, tries an inferential process *(error analysis)* 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. when the knowledge base does not contain the right elements to deduce the root cause, the logical entity starts an *additional data retrieval,* deciding actions devoted to the positioning of new sensors with the aim to collect new diagnostic data. The whole process stops when the Logical Reasoner is able to determine the failure nature. After the Error Analysis has produced the diagnostic inference, the Reasoner can command some opportune actions, in order to fix the discovered fault *(error correction).*

As previously mentioned, the Logical Reasoner acts in a twofold way: through on-line reactive monitoring, or by offline analyses of past network behavior. Since these activities are quite different and use different network representations, we designed a logical framework capable of executing them as distinct tasks. To this end, the Logical Reasoner consists of two functional blocks (see Fig. 4): the On-Line Reasoner (OnLR), devoted to on-line monitoring, and the Off-Line Reasoner (OffLR), capable of performing complex "a posteriori" analyses of network functioning.

3.2.1. On-Line Reasoning

The OnLR is responsible for reactive behavior, and it needs a dynamic representation of network states, achieved by the adoption of the Situation Calculus [5] formalism. The Situation Calculus is exploited to model the network and its dynamic evolution. It is a second order language specifically designed for representing dynamically changing worlds. Such a calculus captures the dynamicity of a system, since it allows the definition of actions, which move the system from a

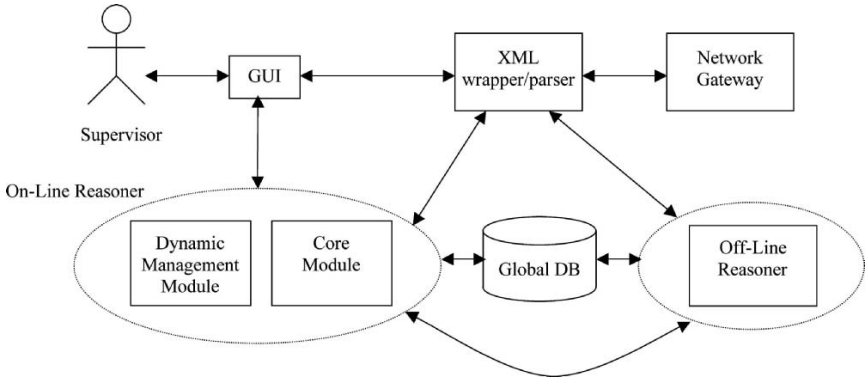

Fig. 4. Logical reasoner functional blocks.

given state to another one. The formalization of the world is performed through well-formed formulas of the first order logic, whilst 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. While the Situation Calculus allows the representation of simple actions, Reactive Golog is an advanced logic programming language, which allows the modeling of more complex behaviors. The adoption of Reactive Golog language is due to its expressiveness and to its capability of providing simple and linear logical frameworks to the programmers.

Reactive Golog is a logic programming language, which inherits the basic elements of Situation Calculus, extending it with procedures and rules. In Reactive Golog there are two types of actions: primitive actions, executed by the system, and exogenous actions, executed by the external world. Therefore, there are two kinds of interaction between the system and the external world: the system changes the world by its actions, and the external world influences the system behavior by exogenous actions. 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." Reactive system behavior is implemented by a concurrent interleaving of a control procedure with a procedure for interrupt management. Conventionally this procedure is called *rules*. OnLR exploits the sensors located in the network nodes to keep its network representation up-to-date, and to collect the events which have already occurred. Using this information, the OnLR focuses its attention on specific network areas and management issues, in order to determine the causes of the observed abnormal behaviors. Since the continuous changes of

network status, the OnLR needs to know and to represent the relevant changes of network features. World evolution is represented by a set of fluents, denoting the truth values of network properties in a given situation. The OnLR, see Fig.4, is composed of a main module (core module) responsible of the reactive functioning and some additional modules devoted to specific tasks. The Core module is a lightweight reactive module that acts as a network event "sentinel," receiving notifications about significant network events. In reply to such events, it performs a basic reasoning activity, collecting further information about network states and, if necessary, delegating specialized logical modules to perform deeper analyses. In order to carry out the above tasks, the OnLR can issue command actions to programmable local agents deployed on network nodes. The OnLR Core module starts its deductive reasoning whenever it detects an abnormal event on the network. Abnormalities reported to the OnLR represent fault conditions and are associated to their local causes by means of the system's distributed monitoring capabilities. The OnLR exploits its logical reasoning mechanism to carry out deeper analyses by means of the activation of additional dynamic management modules. It can thus detect the root causes of the observed symptoms, identifying network faults that, due to their global nature, cannot be signaled by local sensors. For instance, packet losses can be interpreted as possible symptoms of certain network problems, such as network disconnection, routing table corruption, network congestion, or loop detection. In more detail, whilst network disconnection and routing table corruption can be immediately detected by the OnLR Core module using notified data and its general domain knowledge, loop detection is performed by the network itself by means of a distributed coordination technique that exploits local network services programmability, provided at each node.

3.2.2. Off-Line Reasoning

Besides the reactive behavior and the dynamic monitoring of network events carried out by the OnLR modules, it is also necessary to provide a logical entity (the Off-Line Reasoner) capable of performing complex, "a posteriori" analyses of network functioning. This logical entity uses the information distributed at different network nodes and already processed by different system elements (i.e., information passed by the OnLR, or previously deduced and stored in a global database, or logged by nodes in their local databases). To this end, the OffLR is capable both of reconstructing the entire network state for a given interval of time, and of examining the event flow in a limited network area. Moreover, the OffLR can perform a global reasoning process to analyze general network behavior and performance.

The information deduced by means of this higher level analysis can be used, for instance, to detect performance degradation in the communication infrastructure, and to execute a suitable algorithm to improve routing settings.

4. EXPERIMENTAL RESULTS

In order to test and evaluate the logical inference system, we use an experimental cluster of real network nodes, which allows the setup of an experimental test-bed constituted by 40 active nodes equipped with PLAN [22] and ANTS [23] EEs, and super-visioned by the reasoner through the ANGate [8] software package. Each testbed router hosts a PLA capable of tracing all relevant network events, such as status transitions, variable functioning parameters, and traffic flows. In order to study the system reliability and the impact of main architectural parameters, we consider several topologies under three different experimental scenarios. Firstly, we analyze the fault discovering capabilities, considering several trials for each kind of failure that our system can currently manage. In the second scenario, we study the Reasoner capability of undertaking effective reparative actions when network performances degrades. Finally, the third set of experiments are run to test traffic sensor accuracy. Firstly, we carry out experiments dealing with the OnLR. Core capability of anomaly detection and management. The packet losses are considered anomaly symptoms, that activate the logical engine in order to infer the anomaly root causes. For instance, when a packet loss due to TTL expiration occurs, the system tries to discover the existence of a routing loop along the path from the source and destination nodes. To this aim, the OnLR_Core invokes a distributed monitoring service, implemented by means of a simple active capsule that traverses the network from source to destination and stores each visited node. If the capsule reaches an already visited node, the PLA installed on this node executes the code fragment which carries a notification to the OnLR_Core. In order to test fault discovering capability, we measure both the system sensitivity, and specificity. The former is calculated as the percentage ratio between the number of faults detected and the ones generated. The latter, expressing a measure of system capability of avoiding false alarms, can be defined as the ratio between the number of true negatives and the sum of true negatives and false positives. Figure 5(a) shows the experimental results on fault detection. For each kind of failure, we report the percentage of detected cases, lost cases, and false positives, with respect to the total number of faults generated. Experimental results outline that most of the events are detected, thus proving good Reasoner sensitivity. In particular, the figure shows a sensitivity degree of about 88.6% in loop detection, of 85.7% in routing table corruption detection, of 100% in network disconnection detection, of 86.7% in interface failure detection, and of 87.5% in link failure detection. The OnLR also achieves good specificity values (93.5% in disconnection detection, 90.6% in routing table corruption detection, 98.7% in loop detection, 99.1 and 89.5% respectively in link and interface failures detection. From a deeper analysis of experimental data, it arises that false positives and missed fault detections are more frequent during network critical situations, when management information is subject to packet losses and communication failures. Such occurrences claims

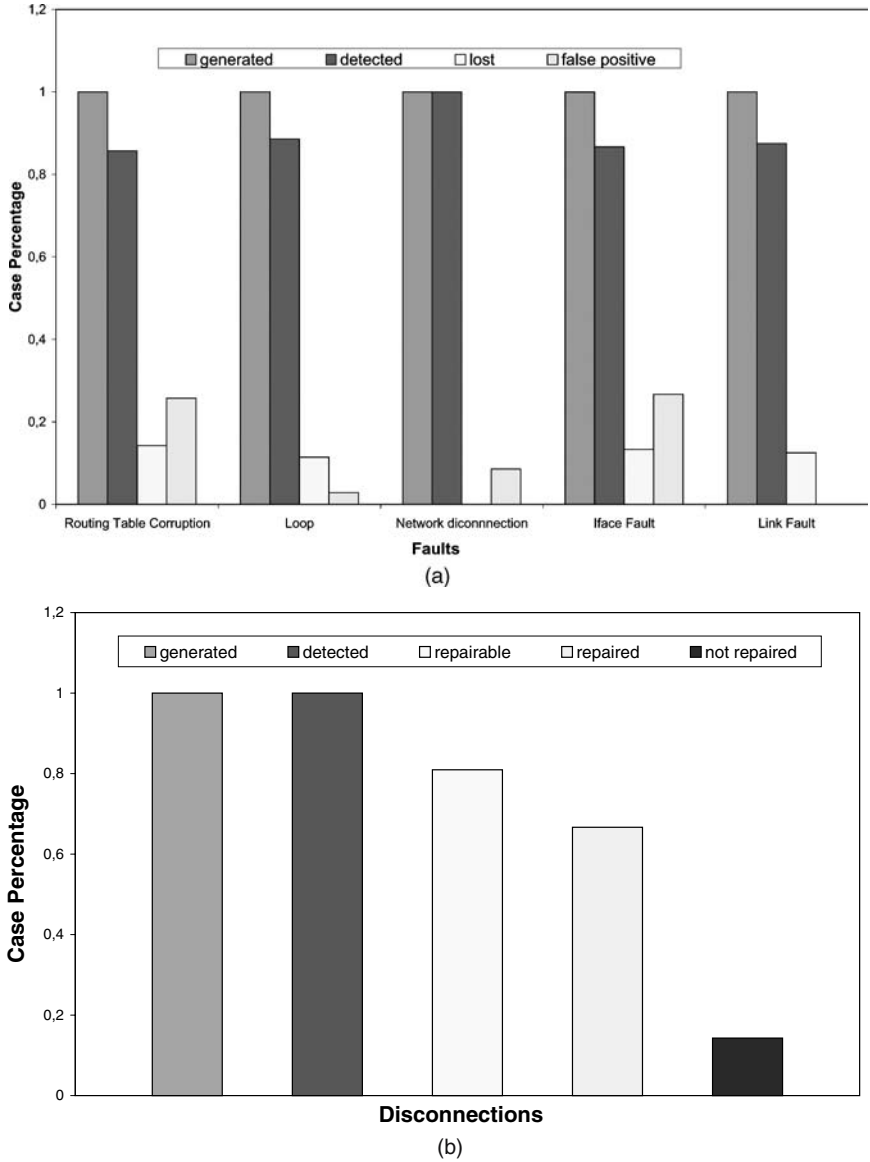

Fig. 5. Fault detection and reparation.

the adoption of a separated network management infrastructure, capable of increasing management system reliability and fault tolerance. In the following set of experiments we analyze the reparative capabilities of the Reasoner towards disconnection occurrences. The system checks the occurrence of a disconnection between any pair of nodes, verifying if the nodes belong to different network "islands." This erroneous situation can be corrected turning on backup links which reconnect the "islands" detected.

Results of these experiments are illustrated in Fig. 5(b), where the percentage of reparable disconnections and of successful repair actions are shown. We can observe that in almost all the cases when the OnLR detects a disconnection, it is also able to undertake a suitable reparative action. In the last experimental scenario, we study the system behavior for traffic monitoring. The OnLR_TrafficMonitor is responsible of traffic sensor tuning on the basis of the network congestion situation. It continuously monitors flows in order to tune activity of sensors, collecting the data which will be used by the OffLR module in order to extract meaningful statistical information about network performances. (Fig. 6 shows a state diagram, representing the OnLR_TrafficMonitor reasoning activity.) In order to measure traffic sensor accuracy, we compare the rate temporal function of generated traffic flows with the estimated instantaneous rate observed by sensors installed on the nodes along the flows. To this end, we use an "ad hoc" network traffic generator, which is able to produce flows according to the traffic models reported in literature [24]. We generate several traffic sessions modeling the rate variation as a step function, to test the sensor accuracy when high rate variations occur. As an example, Fig. 7 reports the instantaneous traffic rate, measured at four randomly chosen network nodes, as a function of time. Experimental results show that flow sensors are able to identify the rate of each different traffic session, and to produce a good estimation of their statistical parameters. High rate observations are characterized by a higher noise component that can be reduced decreasing sensors sampling rate.

5. CONCLUSIONS

This paper proposes an innovative approach for Network Management, complementing a logical reasoner with the versatility of Active Networks and collecting the advantages that come from logical reasoning and network programmability. The approach allows the implementation of a powerful system capable of performing management tasks typically executed by human experts, and of dealing with unusual network situations better than traditional management systems. In order to perform high-level management tasks and to coordinate management activities at the low-level network infrastructure, the inferential engine performs reasoning on a detailed ontological model capable of describing as better as possible networking concepts. The logical reasoner is able to deduce knowledge and find

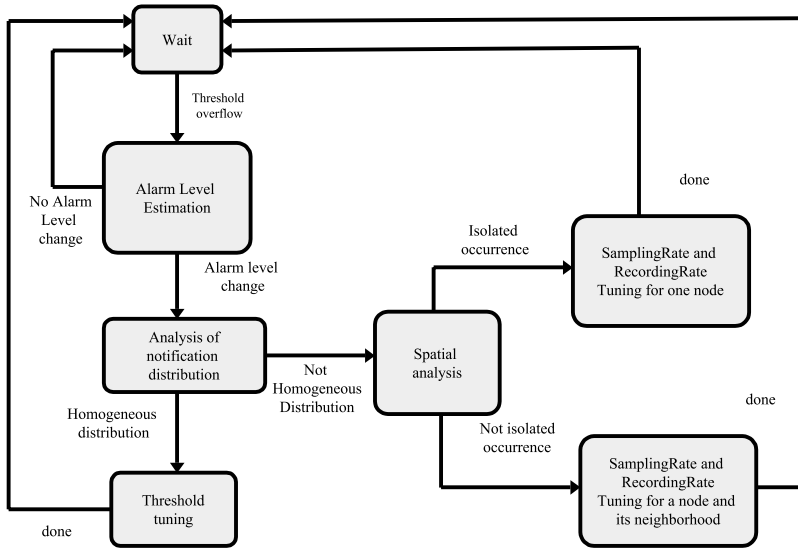

Fig. 6. State diagram of the OnLR_TrafficMonitor reasoning.

correlations from data and events which are distributed on different network areas and which occur 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

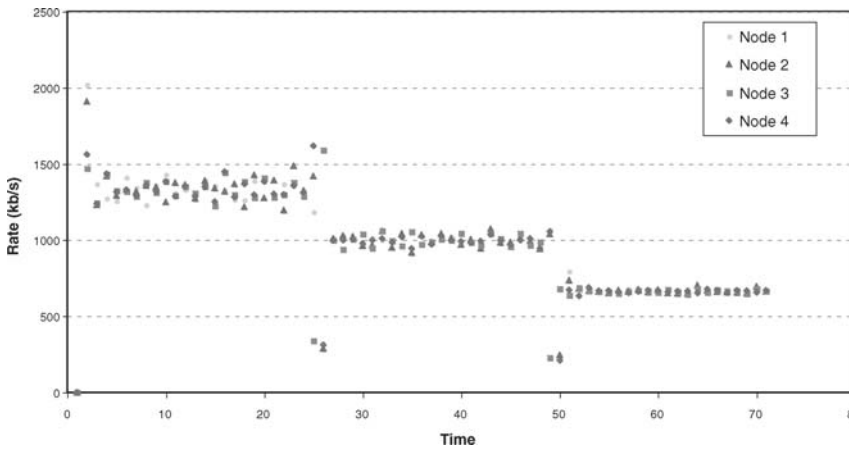

Fig. 7. Traffic monitoring measurements.

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. W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2,* 3rd edn., Addison-Wesley, Reading, MA, 2003.
2. L. Raman, OSI Systems and Network Management, *IEEE Communications Magazine*, Vol. 36, No. 3, pp. 46–53, 1998.
3. D. L. Tennenhouse and D. J. Wetherall, Towards an Active Network architecture, *ACM Computer Communication Review*, Vol. 26, No. 2, pp. 15–20, 1996.
4. D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minde, Survey of Active Network research, *IEEE Communications Magazine*, Vol. 35, No. 1, 1997.
5. J. McCarthy, Situations, actions and causal laws, in M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, MA, pp. 410–417, 1968.
6. R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, Cambridge, MA, 2001.
7. S. Mazumdar and A. A. Lazar, Objective-Driven Monitoring for Broadband Networks, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 3, 1996.
8. A. Barone, P. Chirco, G. Di Fatta, and G. Lo Re, A management architecture for Active Networks, *Proceedings of IEEE International Workshop on Active Middleware Sendees*, Edinburgh, UK, pp. 41–48, July 2002.
9. Y. Yemini, A. V. Konstantinou, and D. Florissi, NESTOR: An Architecture for Self-Management and Organization, *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 5, 2000.
10. A. W. Jackson, J. P. G. Sterbenz, M. N. Condell, and R. R. Hain, Active Network monitoring and control: The SENCOMM architecture and implementation, *Proceedings of IEEE DARPA Active Networks Conference and Exposition*, San Francisco, CA, May 2002.
11. D. Raz and Y. Shavitt, *An Active Network Approach for Efficient Network Management*, number 1653 in LNCS, Springer-Verlag, Berlin, 1999.
12. L. Ricciulli, P. Porras, P. Lincoln, P. Kakkar, and S. Dawson, An adaptable network control and reporting system (ANCORS), *Proceedings of IEEE DARPA Active Networks Conference and Exposition*, San Francisco, CA, May 2002.
13. S. F. Bush and A. Kulkarni, *Active Networks and Active Network Management: A Proactive Management Framework*, Kluwer, Dordretch, 2001.
14. S. F. Bush, Active virtual network management prediction: Complexity as a framework for prediction, optimization, and assurance, *Proceedings of IEEE DARPA Active Networks Conference and Exposition*, San Francisco, CA, May 2002, pp. 534–553.
15. E. Al Shaer, Active Management Framework for Distributed Multimedia Systems, *Journal of Network and Systems Management*, Vol. 8, No. 1, pp. 49–72, 2000.
16. D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, A knowledge plane for the Internet, *Proceedings of ACM SIGCOMM*, pp. 3–10, August 2003.
17. M. Wawrzoniak, L. L. Peterson, and T. Roscoe, Sophia: An Information Plane for Networked Systems, *ACM Computer Communication Review*, Vol. 34, No. 1, pp. 15–20, 2004.
18. N. J. Nilson, *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann, San Fransisco, CA, 1998.

19. T. R. Gruber, A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, Vol. 5, No. 2, pp. 199–220, 1993.
20. G. Antoniou and F. van Harmelen, Web Ontology Language: OWL, in S. Staab and R. Studer (eds.), *The Handbook on Ontologies in Information Systems*, Springer-Verlag, Berlin, 2003.
21. L. Kerschberg, R. Baum, A. Waisanen, I. Huang, and J. Yoon, Managing faults in telecommunications networks: A taxonomy to knowledge-based approaches, *Proceedings of IEEE International Conference on Systems*, Man and Cybernetics, pp. 779–784, 1991.
22. M. Hicks, P. Kakkar, J. T. Moore, C. Gunter, and S. Nettles, PLAN: A Packet Language for Active Networks, *Proceedings of ACM International Conference on Functional Programming*, pp. 86–93, September 1998.
23. D. J. Wetherall, J. Guttag, and D. L. Tennehouse, ANTS: A toolkit for building and dinamically deploying network protocols, *Proceedings of IEEE OPENARCH*, San Francisco, CA, April 1998.
24. Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, On the characteristics and origins of Internet flow rates, *Proceedings of ACM SIGCOMM*, August 2002.

Salvatore Gaglio graduated in Electronic Engineering from University of Genoa, Italy, in 1977. He is a professor of computer science and artificial intelligence at the University of Palermo, Italy. His present research activities are in the areas of artificial intelligence and robotics. He is a member of IEEE, ACM, and AAAI.

Luca Gatani received the Laurea degree in Computer Engineering, in 2003, from University of Palermo, Italy, where he is currently working for a PhD degree in Computer Engineering. His research interests include peer-to-peer systems, network management, multicast transmissions, and wireless sensor networks.

Giuseppe Lo Re received the Laurea degree in Computer Science from University of Pisa in 1990, Italy, and PhD in Computer Engineering from University of Palermo, Italy in 1999. Currently he is an associate professor at the University of Palermo. His research interests are in the area of computer communication networks and distributed systems. He is a member of IEEE Communication Society and ACM.

Alfonso Urso received the Laurea degree in Electronic Engineering and the Ph.D in Systems Engineering from University of Palermo, Italy, in 1992 and 1997, respectively. In 2000, he joined the Italian National Research Council (CNR) where, currently, he is a Researcher in systems and computer engineering. His current research interests are in the area of computer networks and artificial intelligence.