Distributed Management of High-layer Protocols and Network Services Through a Programmable Agent-based Architecture

Luciano Paschoal Gaspary, Luis Felipe Balbinot, Roberto Storch, Fabricio Wendt, and Liane Rockenbach Tarouco

Federal University of Rio Grande do Sul, Institute of Informatics,
Av. Bento Gonçalves, 9500 – Agronomia – CEP 91591-970 – Porto Alegre, Brazil,
paschoal@inf.ufrgs.br,
WWW home page: http://www.inf.ufrgs.br/~paschoal

Abstract. This paper proposes an architecture for distributed management of upper layer protocols and network services called **Trace**. Based on the IETF Script MIB, the architecture provides mechanisms for the delegation of management tasks to mid-level managers, which interact with monitoring and action agents to have them executed. The paper introduces PTSL (*Protocol Trace Specification Language*), a graphical/textual language created to allow network managers to specify protocol traces. The specifications are used by mid-level managers to program the monitoring agents. Once programmed, these agents start to monitor the occurrence of the traces. The information obtained is analyzed by the mid-level managers, which may ask action agents for the execution of procedures (Perl scripts), making the automation of several management tasks possible.

1 Introduction

The use of computer networks to support a growing number of businesses and critical applications has stimulated the search for new management solutions that maintain not only the physical infrastructure, but also the protocols and services that flow over it. The popularization of electronic commerce (e-commerce) and the increasing use of this business modality by companies, for instance, imply using the network to exchange critical data from the organization and from its customers. Protocols and services that support these applications are critical and, therefore, need to be carefully monitored and managed.

Not only critical applications require special attention. New protocols are frequently released to the market to support an increasing set of specific functionalities. These protocols are quickly adopted by network users. As a result of this fast proliferation, weakly-tested and even faulty protocols are disseminated to the network consuming community. In several cases these anomalies, as well as the miscalculated use of resources, are the cause of network performance degradation and end up unnoticed.

We believe that most of the research carried on so far try to provide mechanisms to guarantee higher availability and performance for networks (e.g. Hood and Ji work on proactive fault detection [2]). While solutions to manage physical network infrastructure are established and tested, it is still needed to investigate ways to provide effective management of applications and protocols.

Existing management tools are not completely prepared to allow the monitoring of these new applications and protocols. Most of the tools only allow the monitoring of a closed set of them. The ability to observe new ones depends on the firmware update of the monitoring hardware (e.g. RMON2 probes [3]) or on the programming in low level languages as the extensible probe architecture proposed by Malan and Jahanian [4]. Due to the complexity of the task, most network managers neglect this possibility.

In some approaches it is possible to recognize and count packet flows specified by simple filtering rules (e.g. `tcpdump`-like filters used by `ntop` [5]) or by descriptive languages such as SRL [6], used by `NeTraMet` [7]. However, these filtering languages lack constructors that allow a rule to be defined as a sequence of packets each with specific filtering options, making it impossible to accomplish time-based or correlated analysis of flows.

Other solutions such as Tivoli Enterprise [8] are intrusive, since they require that developers insert specific monitoring procedure calls while developing applications. This approach is only suitable for applications developed in-house. It cannot be used to manage proprietary protocols (e.g. web browsers and servers and e-mail client and servers). Besides that, one must invest on personnel training to use the monitoring APIs.

Regarding the type of information gathered by monitoring engines, some approaches such as the IETF RMON2 MIB (Remote Network Monitoring Management Information Base version 2) store, for a pre-defined set of high-layer protocols supported by the probe, the number of packets sent/received by a host or exchanged by host pairs. Gaspary et al. describe in [9, 10] the advantages and limitations of the RMON2 MIB. One of the RMON2 weaknesses is that it does not store any information related to performance, but it has been discussed by the Remote Network Monitoring group at the IETF [11].

Finally, we should point out that many management tools are limited to monitoring [5, 7] and the network manager has to take actions manually when unexpected behaviors from these protocols are observed.

In this paper we present **Trace**, an architecture for distributed management of enterprise networked applications, high-layer protocols and network services [12] based on the IETF Script MIB [13]. Through a graphical and textual language based on finite state machines, the network manager defines protocol traces to be observed. These specifications are readily received by one or more programmable agents that immediately start to check whether a defined trace occurs or not. The observation of these traces in the network traffic triggers actions, which are also determined by the network manager.

The paper is organized as follows: section 2 describes the language to specify protocol traces. In section 3 the architecture is presented. Section 4 illustrates

how to accomplish fault management using the architecture. In section 5 we present a summary and concluding remarks.

2 Protocol Trace Representation Using PTSL

In this section we propose PTSL (*Protocol Trace Specification Language*), a graphical and textual language for the representation of high-layer protocol traces. The languages are not equal. The textual one makes the complete representation of a trace possible, including the specification of both the state machine and the events that trigger the transitions. On the other hand, by using the graphical language one can graphically represent the state machine but only label the events that trigger the transitions.

2.1 Organization of a Specification

The textual specification of a trace begins with the keyword `Trace` and ends with `EndTrace`. Initially, the manager may describe some optional items to the specification (see figure 1, lines 2–7). Next, it is broken down into three sections: `MessagesSection` (lines 8–10), `GroupsSection` (lines 11–13) and `StatesSection` (lines 14–16), where messages to be observed, grouping and state machines that describe the trace are respectively specified.

```
1  Trace "Successful WWW access"
2  Version: 1.0
3  Description: WWW access with 200 response.
4  Key: HTTP, 200, OK
5  Port: 80
6  Owner: Luciano Paschoal Gaspary
7  Last Update: Fri, 23 Sep 2000 15:15:03 GMT
8  MessagesSection
9  ...
10 EndMessagesSection
11 GroupsSection
12 ...
13 EndGroupsSection
14 StatesSection
15 ...
16 EndStatesSection
17 EndTrace
```

Fig. 1. Schematic representation of a textual specification.

If the trace to be monitored belongs to a single application-layer protocol then the network manager may specify the TCP or UDP port number using the `Port` parameter (line 5). It will simplify packet classification during the monitoring phase.

2.2 State Machines

The trace of a protocol is defined through a finite state machine. The network manager may define a model to monitor just a part of or the whole protocol, or interactions that comprehend more than one protocol. Figure 2 shows two trace examples. In the first example (a), the manager is interested in monitoring the successful accesses to a WWW server. The trace shown in (b) does not describe a single protocol; it is rather made up of a name resolution request (DNS protocol), followed by an ICMP `Port Unreachable` message. This trace occurs when the host where the service resides is on, but the *named* daemon is not running.

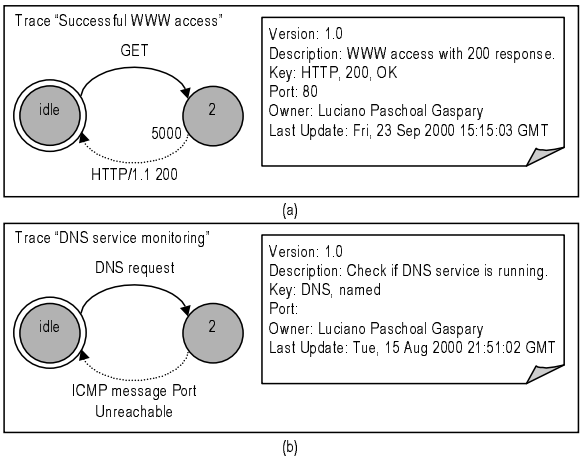

Fig. 2. Graphical representation of a trace. (a) Successful WWW request. (b) DNS request not replied because *named* daemon is not executing.

As one can see states are represented by circles. The initial state has the label `idle` associated to it. The final state is represented by two concentric circles. In both examples the initial and final states are the same (`idle`). Transitions are represented by unidirectional arrows. A continuous arrow indicates that the transition is triggered by the client host, whereas a dotted arrow denotes that it is caused by the server host. The text associated to a transition only labels the event (specified as a message or grouping in the textual language) that will trigger it. It means that the whole specification of a transition only can be done using the textual language. The graphical representation of the state machines shown in figure 2 can be mapped to the textual specification presented in figure 3.

2.3 Transitions

In addition to making a high-level representation of traces, it is necessary to describe what causes the change of states. Before describing the adopted so-

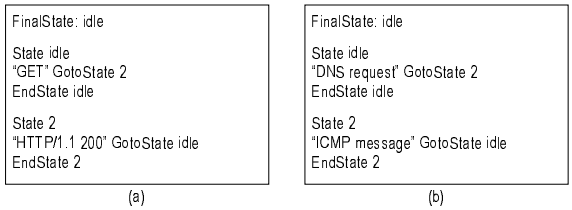

Fig. 3. Textual representation of state machines.

lution, it is important to highlight that high-layer protocols are specified in many different ways. Larmouth classifies them as character or binary-based [14]. Character-based protocols are defined as a set of text lines coded in ASCII (e.g. HTTP and SMTP). Binary protocols, on the other hand, are defined as strings of octets or bits (e.g. TCP).

Considering the differences between both protocol types, we propose state transitions to be represented by a positional approach. Taking the example shown in figure 2a, we present (see figure 4a) how to represent the transition `HTTP/1.1 200`.

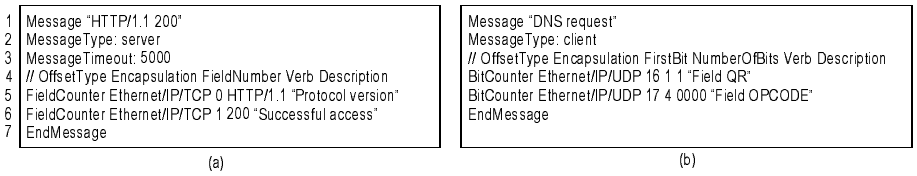

Fig. 4. Representation of (a) Character-based and (b) Binary protocol fields.

As the transition is expected to be triggered by the server host, one must set the `MessageType` field to `server` (line 2). Since both protocol fields (`HTTP/1.1` and `200`) belong to a character-based protocol, the search for their positions within the packets is made by fields (`FieldCounter`, lines 5–6). In this example, `HTTP/1.1` is the first string that appears on the message and therefore its offset is 0 (third parameter in line 5). The second string to appear is `200` and its offset is 1 (line 6). For each protocol field defined in a message it is also necessary to inform where to look for it (encapsulation `Ethernet/IP/TCP`, lines 5–6).

When the transition is caused by a binary protocol, the offset is presented in bits (`BitCounter`). In this case, it is necessary to inform where the field starts (`FirstBit`) and the number of bits to be observed from this offset on (`NumberOfBits`). A standard DNS request can be recognized by two fields: `QR` (when set to `1` indicates a request to the server) and `OPCODE` (when set to `0` represents a standard query). Field `QR` is 16 bits away from the beginning of the header and its size is 1 bit. Field `OPCODE` starts in the seventeenth bit and

occupies 4 bits. In figure 4b the textual representation of a standard DNS request is shown.

It is possible to group one or more messages into one single transition. For example, in figure 2a it would be possible to replace the `HTTP/1.1 200` with the grouping `HTTP/1.1 2XX`. In this case the trace would monitor the rate of all successful WWW operations generated by client requests (`2XX`) instead of only observing the occurrence of WWW accesses whose return code is 200 (successful request). Figure 5 shows the representation of this grouping (lines 16–18).

```
1  MessagesSection
2  ...
3  Message "HTTP/1.1 200"
4  MessageType: server
5  FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Protocol Version"
6  FieldCounter Ethernet/IP/TCP 1 200 "Successful request"
7  EndMessage
8  Message "HTTP/1.1 202"
9  MessageType: server
10 FieldCounter Ethernet/IP/TCP 0 HTTP/1.1 "Protocol Version"
11 FieldCounter Ethernet/IP/TCP 1 202 "Request accepted but not processed"
12 EndMessage
13 ...
14 EndMessagesSection
15 GroupsSection
16 Group "HTTP/1.1 2XX"
17 Messages: "HTTP/1.1 200", "HTTP/1.1 202", ...
18 EndGroup
19 EndGroupsSection
20 StatesSection
21 FinalState: idle
22 State idle
23 "GET" GotoState 2
24 EndState idle
25 State 2
26 "HTTP/1.1 2XX" GotoState idle
27 EndState 2
28 EndStatesSection
```

Fig. 5. Representation of message grouping.

In some cases the network manager may be interested in observing the occurrence of a certain string within the data field of a certain protocol, no matter where it is located. To do that, in the definition of such a message one must use `NoOffset` as the `OffsetType` parameter. This feature is interesting, for instance, to observe the attempt of an intrusion. The example presented in figure 6 defines that every TCP packet must be tested for the occurrence of the string `/etc/passwd` (line 4).

We have also created a mechanism to allow the determination of a timeout to a transition to occur. To do that one must associate a timeout value (in

```
1 Message "/etc/passwd"
2 MessageType: client
3 // OffsetType Encapsulation Verb
4 NoOffset Ethernet/IP/TCP /etc/passwd
5 EndMessage
```

Fig. 6. Non-specified offset message field.

milliseconds) to the message definition (see figure 4a, line 3). When not defined, a default value is used by the network monitor.

3 The Trace Architecture

The architecture we propose is an extension of the existing distributed management infrastructure standardized by the IETF [15] with high-layer protocol and network service management capabilities. Figure 7 shows the main components of the architecture. It is composed of management stations, mid-level managers, programmable monitoring agents and programmable action agents. The following sub-sections describe the components of the architecture and their interactions with each other.

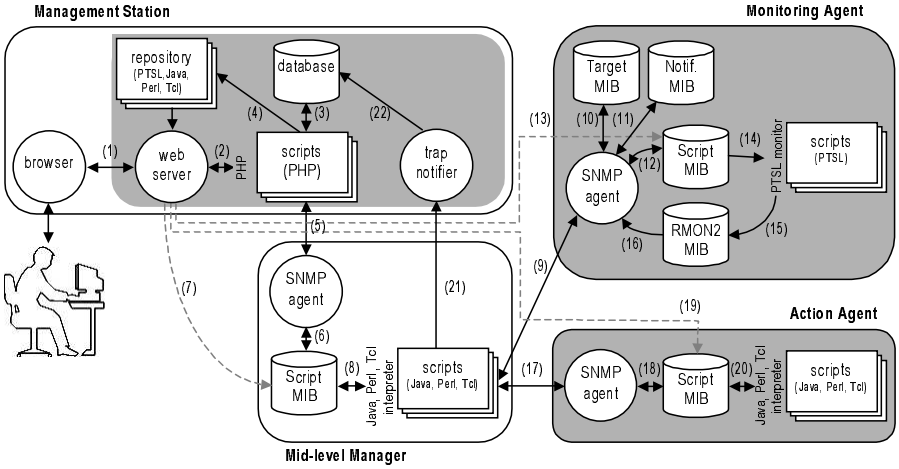

Fig. 7. Components of the architecture.

3.1 Management Station

The most important activities accomplished by the network manager from a management station are (a) registration of mid-level managers, monitoring and

action agents, (b) specification of protocol traces and actions, (c) specification, delegation, observation and interruption of management tasks and (d) receipt and visualization of traps.

As the whole architecture is based on the Script MIB, protocol traces, actions and management tasks are scripts executed by monitoring agents, action agents and mid-level managers, respectively. Protocol traces are specified by the network manager using the PTSL language. Actions are scripts developed using Java or any scripting language such as Tcl and Perl. Management tasks may also be implemented using any language and coordinate monitoring and action agents. Such a script programs the monitoring agents, observes the occurrence of the trace and activates action agents when a condition associated to a protocol trace holds. The same script may also report events to the management station raising traps.

At the management station the network manager can specify traces using a graphical tool (see an example of such a tool in figure 8) or, if he knows the language, by editing a text file. The same occurs with actions and management tasks. The specification of protocol traces, actions and management tasks are stored in the database (figure 7, see flows (1, 2, 3) in diagram). When a management task is about to be delegated, they are mapped to files and stored in the repository (4).

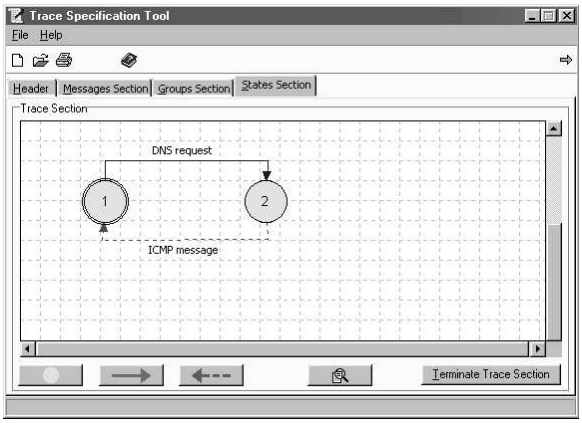

Fig. 8. Prototype of the tool for trace specification.

Communication between the management station and the mid-level managers takes place using the SNMP protocol (Script MIB) (5, 6). The manager can delegate a management task to a mid-level manager as well as abort it at any time. Intermediate and final results of the execution of a management task are stored directly at the Script MIB of the mid-level manager responsible for the task and can be retrieved by the management station using the SNMP protocol (5, 6).

The manager may receive traps through an element called *trap notifier* (21). When received, all traps are stored in a database (22). The traps are permanently retrieved by a script (3) that updates the manager's web browser (2, 1) using the push technology.

3.2 Mid-level Manager

Mid-level managers execute and monitor management tasks delegated by the management station and report the most important events to it. The number of mid-level managers is determined by the network manager and depends on the size and complexity of the infrastructure to be managed.

The process of configuring mid-level managers is the following: the network manager defines a management task and stores it at the repository (1, 2, 3, 4). Next, the activation of the task must be scheduled using the Script MIB (5, 6). In order to do that, the mid-level manager has to be informed about the location of the task (script). When activated, the task is retrieved from the repository using the HTTP protocol (7) and executed (8).

The script executed by the mid-level manager installs the protocol trace (9, 12) and the action script (17, 18), requests the monitoring agent to start observing the occurrence of the protocol trace just installed (9, 12), polls RMON2 variables periodically to monitor the occurrence of the trace (9, 16) and, depending on what is observed, dispatches the execution of the action script (17, 18) or raises a trap to the manager (21). The script communicates with the agents using the SNMP protocol.

The same script may also subscribe at monitoring and action agents to the traps it wants to receive. The Target MIB is used to identify the management task (IP address and UDP port) (9, 10). Using the Notification MIB the mid-level manager indicates (through filters) which traps should be sent to the management task, whose location was identified at the Target MIB (9, 11) [16]. When the script receives a trap, it may dispatch the execution of an action (17, 18) or correlate it with previously received traps. The result of these operations may be informed to the management station (21).

3.3 Monitoring Agent

The monitoring agents are responsible for observing the traffic on the network segment where they are installed. They are configured by mid-level managers and are called programmable because they are able to monitor protocol traces delegated dynamically by the network manager. This flexibility is obtained through the language presented in section 2. When the mid-level manager sets the monitoring agent up (9, 12), the former defines which protocol trace it should retrieve (it is indicated within the script that implements the task). Once retrieved (13), the trace file is loaded by the monitoring engine and the observation starts (14).

Whenever the occurrence of the trace is observed between any pair of hosts, information is stored within an RMON2-like MIB (15). This MIB is different from the standard because the `protocolDir` group is writable in our approach.

Therefore the probe stores statistics according to the protocol traces of interest to the network manager. Additionally, the granularity of the monitoring becomes higher. Instead of storing overall statistics on traffic generated by a given protocol, statistics are generated according to the occurrence of specified traces or transactions.

The `alMatrix` group from RMON2 MIB stores statistics on the trace when the latter is observed between every pair of hosts. Table 1 shows the contents of the `alMatrixSD` table. It gathers the observed number of packets and octets exchanged between every pair of hosts (client/server) using the protocol traces being monitored by the probe. In the example, two traces were observed: `Successful WWW access` and `DNS service monitoring` (previously shown in figure 2a and b).

Table 1. Information obtained by referring to the `alMatrixSD` table.

Source Address	Destination Address	Protocol	Packets	Octets
17.16.10.1	17.16.10.2	Successful WWW access	254	120.212
17.16.10.6	20.24.20.2	Successful WWW access	20	10.543
17.16.10.1	17.16.10.33	DNS service monitoring	4	4.350
17.16.10.32	17.16.10.33	DNS service monitoring	8	7.300

The disadvantage of using RMON2 MIB is that it does not have objects capable of storing information related to performance. For this reason, our group is currently considering the possibility of using, in addition to that, an RMON2 extension, such as Application Performance Measurement MIB [11]. Table 2 shows the kind of information stored by this MIB. The first line shows that the trace `Successful WWW access` has been observed 127 times between hosts 17.16.10.12 and 17.16.10.2. Additionally, the mean response time was 6 seconds.

Table 2. MIB with information on performance.

Client	Server	Protocol	Success.	Unsuccess.	Responsiv.
17.16.10.12	17.16.10.2	Successful WWW access	127	232	6 sec.
17.16.10.12	20.24.20.2	Successful WWW access	232	112	17 sec.
17.16.10.1	17.16.10.33	DNS service monitoring	2	0	3 sec.

3.4 Action Agent

Action agents reside in hosts where network services are executed. Their function is to perform a given operation on these services. Let us take as example the

DNS service. Figure 2b shows a trace that enables to detect when the *named* daemon is not in execution. There may be a management task, delegated to the mid-level manager, that monitors the occurrence of this trace. If that is the case, the action to be taken is to contact the action agent (see flow (17) in figure 7), which is located in the host where the DNS service is installed, and request the execution of a script to restart the daemon (18, 19, 20) (see the example of such a script developed in Perl in figure 9). The result obtained by the action agent is accessible to the mid-level manager (17, 18), which may send it to the manager for notification purposes (21).

```
#!/usr/bin/perl

my $pid;

# Verify if the process named is executing.
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
}

# If named is running, restart it using a HUP signal, otherwise
# instantiate the process again.
if (defined $pid) {
   print "Restarting named (sending HUP signal)...\n";
   `/bin/kill -HUP $pid`;
} else {
   print "Starting named (was not running)...\n";
   `/usr/sbin/named &`;
}

# Test if the process is executing.
if (-e "/var/run/named.pid") {
   $pid = `/bin/cat /var/run/named.pid`;
   print "The named daemon is up and running as PID $pid\n";
} else {
   print "The named daemon could not be started!\n";
}
```

Fig. 9. Perl script to restart the *named* daemon.

4 Fault Management of Network Services

Fault management is an important target of the proposed architecture. An example of fault management concerning to high-layer protocols and network services is checking the availability of a network service and restart it if it is not running. Figure 10 shows how this can be achieved using the architecture. In this case the task delegated to the mid-level manager is supposed to monitor the DNS service.

This monitoring is performed by sniffing the packets seen in the segment. In a situation where the daemon responsible for the DNS service is not running, the agent will observe a DNS request and some time later a `Port Unreachable` ICMP message from the serving host. In this case the mid-level manager should contact an action agent, which resides in the DNS serving host and request the

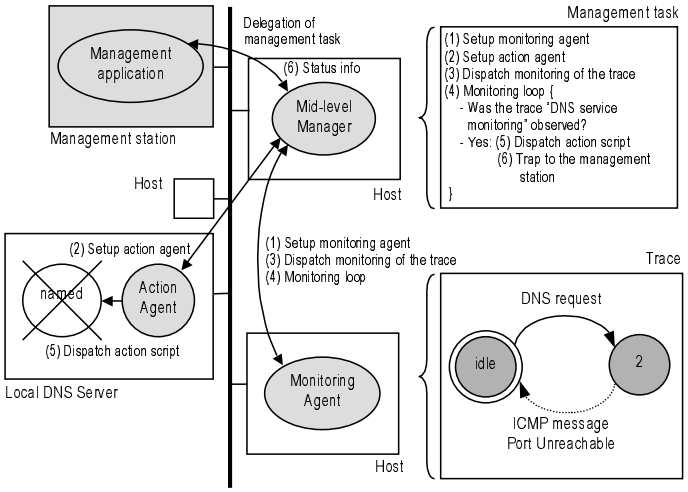

Fig. 10. Execution steps of a management task.

execution of a script to restart the daemon (such as the one illustrated in figure 9). The textual specification of the trace `DNS service monitoring` is shown in figure 11.

The architecture was designed to take into account all the standard functional areas of management: fault, configuration, accounting, performance and security (FCAPS). Our research group has explored in [12] its characteristics to validate the usefulness of the architecture for the management of high-layer protocols and network services.

5 Conclusions

This work presented a distributed architecture for the management of high-layer protocols and network services based on programmable agents. Motivated by the growing need companies have to monitor high-layer protocols and their critical applications, the work proposes a flexible architecture able to follow the fast dissemination of protocols and networked applications (that need to be managed). The architecture may be used either in corporate networks or in application service providers.

The most important contribution of the architecture is the granularity of the monitoring. The observation of network traffic on a transaction basis makes the understanding of protocol and networked application behaviors possible. The language proposed to specify protocol traces is simple, but the network manager has to know the format of the packets exchanged by the application or protocol to be managed.

Another significant contribution of the architecture is the possibility to do more than just monitoring. Management tasks provide the manager with mecha-

```
Trace "DNS service monitoring"

Version: 1.0
Description: Trace to detect  when named is not running
Key: named, fault, DNS service
Port:
Owner: Luciano Paschoal Gaspary
Last Update: Tue, 10 Aug 2000 15:30:58 GMT

MessagesSection

Message "DNS Request"
// See code in figure 4b.
EndMessage

Message "ICMP Message"
MessageType: server
// OffsetType Encapsulation FirstBit NumberOfBits Verb Description
BitCounter Ethernet/IP 0 8 00000011 "Type field=00000011?"
BitCounter Ethernet/IP 8 8 00000011 "Code field=00000011?"
EndMessage

EndMessagesSection

StatesSection

// See code in figure 3b.

EndStatesSection

EndTrace
```

Fig. 11. Trace to monitor the DNS service.

nisms to monitor the occurrence of protocol traces and to dispatch management scripts executed by programmable action agents. These mechanisms contribute to management automation in some scenarios.

As described by Strauß[17], "distributed management in general [18] and the Script MIB specifically are expected to bring various advantages over the centralized concept suited for the raising demands in network management. A commonly mentioned advantage is the increased scalability due to the delegation of management tasks from the centralized network management station to mid-level managers. This implies that CPU and network load is also delegated to the subnets to which the mid-level managers belong. Another major advantage is concerned with the robustness of management tasks. While centralized management systems require a reliable network, the distributed approach allows to delegate some sensible manager functions next to the observed agents. Hence these functions may become independent from less reliable WAN links, for example".

The architecture requires more work to be controlled than a single centralized management system. The management of its components becomes more complicated. It is necessary to distribute and update scripts, control running scripts, gather and correlate intermediate and final results. We believe that such operations, as well as the specification of traces, can be simplified by adding an easy-to-use interface to the management application. Currently, our research group is working on the improvement of the prototype. After that, a larger scale validation will be done.

References

1. L. L. Ho, C. J. Macey, and R. Hiller. "A Distributed and Reliable Platform for Adaptive Anomaly Detection in IP Networks". In *Proc. 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, pp. 33–46.
2. C. S. Hood and C. Ji. "Intelligent Agents for Proactive Fault Detection". *IEEE Internet Computing*, vol. 2, no. 2, pp. 65–72, March/April 1998.
3. S. Waldbusser. "Remote Network Monitoring Management Information Base Version 2 using SMIv2". Request for Comments 2021, January 1997.
4. G. Malan and F. Jahanian. "An Extensible Probe Architecture for Network Protocol Performance Measurement". In *Proc. SIGCOMM*, Vancouver, September 1998.
5. L. Deri and S. Suin. "Ntop: Beyond Ping and Traceroute". In *Proc. of the 10th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, p. 271–283.
6. N. Brownlee. "SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups". Request for Comments 2723, October 1999.
7. NeTraMet. Available at http://www.auckland.ac.nz/net/Internet/rtfm/.
8. C. Cook et al. "An Introduction to Tivoli Enterprise". First edition. USA: International Technical Support Organization, 1999. Available at http://www.redbooks.ibm.com.
9. L. P. Gaspary and L. R. Tarouco. "Characterization and Measurements of Enterprise Network Traffic with RMON2". In *Proc. 10th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Zurich, October 1999, pp. 229–242.
10. L. P. Gaspary and L. R. Tarouco. "Managing Users, Applications and Resources with RMON2". In *Proc. Global Telecommunications Conference, Symposium on Enterprise Applications and Services*, Rio de Janeiro, December 1999, pp. 1997–2001.
11. S. Waldbusser. "Application Performance Measurement MIB". Internet Draft, November 2000.
12. L. P. Gaspary, L. F. Balbinot, R. Storch, F. Wendt, and L. R. Tarouco. "Towards a Programmable Agent-based Architecture for Enterprise Application and Service Management". In *Proc. First IEEE/IEC Enterprise Networking Applications and Services Conference*, Atlanta, June 2001.
13. D. Levi and J. Schönwälder. "Definitions of Managed Objects for the Delegation of Management Scripts". Internet Draft, Nortel Networks, TU Braunschweig, July 2000.
14. J. Larmouth. *ASN.1 Complete*. Open Systems Solutions, 1999.
15. J. Schönwälder, J. Quittek, and C. Kappler. "Building Distributed Management Applications with the IETF Script MIB". *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 5, pp. 702–714, 2000.
16. D. Levi, P. Meyer, and B. Stewart. "SNMP Applications". Request for Comments 2573, April 1999.
17. F. Strauß. "Advantages and Disadvantages of the Script MIB Infrastructure". Project Report, October 2000. Available at http://www.ibr.cs.tu-bs.de/projects/jasmin/.
18. J. Schönwälder. "Network Management by Delegation – From Research Prototypes Towards Standards". *Computer Networks and ISDN Systems*, vol. 29, no. 15, pp. 1843–1852, November 1997.