RETRIEVING CALL DETAIL RECORDS FROM ASTERISK USING SNMP

Cristian Mihai VANCEA Virgil DOBROTA
Technical University of Cluj-Napoca, Communications Department
26-28 George Baritiu Street, 400027 Cluj-Napoca, Tel: +40-264-401226, Fax: +40-264-597083
E-mails: {Mihai.Vancea,Virgil.Dobrota}@com.utcluj.ro

Abstract: Several SNMP agents could be integrated by IETF's AgentX protocol which separates the communication with SNMP manager in two parts: AgentX between sub-agents and master agent, and SNMP between master agent and manager. The paper extends the SNMP software agent of the IP-based PBX Asterisk. The management information related to Application Layer (calls and charging) are collected in .log files, parsered and then stored into .xml files. Being an alternative to legacy commercial implementations, this solution has the advantage of using the SNMP, a generalized non-proprietary management protocol. This could be a step forward towards an integrated management where the access, distribution and core networks, together with applications are using the same mechanism which will not be external anymore as it is nowadays.

Key words: *agent, Asterisk, management information base, manager, SNMP*

I. INTRODUCTION

The primary goal of the network management model created by ISO is the understanding of the major management system functions. It consists of five conceptual areas to act as a guideline in practice known as FCAPS. This acronym stands from *F*ault, *C*onfiguration, *A*ccounting, *P*erformance and *S*ecurity. However this paper is focused on the third issue only, i.e. the accounting. We selected the IETF's SNMP (Simple Network Management Protocol), running at the Application Layer and defining four components: a) NMS (Network Management Station), known also as the manager; b) the managed nodes, each of them containing a processing entity called agent, that reports information (responses and notifications) to the manager; c) the network management protocol; d) the management information specified by the SMI (Structure of Management Information) and by the collection of managed objects called MIB (Management Information Base) [1]. The selected platform for the testbed demonstrator was Asterisk, the world's leading open source PBX based on IP. It was released under the GNU GPL (General Public License) and it supports a wide range of protocols for the handling and transmission of voice over traditional telephony interfaces including H.323, SIP (Session Initiation Protocol), MGCP (Media Gateway Control Protocol), and SCCP (Skinny Client Control Protocol) [2]. Furthermore using its proprietary IAX (Inter-Asterisk Exchange) protocol, Asterisk merges voice and data traffic seamlessly across disparate networks.

The paper is organized as follows. The second section discusses the design principles based on a multi-agent framework where the agents actually may contact the manager through a master agent only. The third paragraph offers details about the proposed MIB and it explains its integration with the existing collection of managed objects. This involves additional software to be installed, following a newly defined *parserd* algorithm. Section four is focused on experimental results based on a testbed with Asterisk running SNMP agent, NMS running a newly defined SNMP Manager and several VoIP phones. Although this paper appears to be focused on technical issues, its intention was to contribute at conceptual level too, by demonstrating that the SNMP could be used as a generalized protocol for both networks and applications. Conclusions and future work are included in the last paragraph.

II. DESIGN PRINCIPLES

For activating SNMP in Asterisk, Net-SNMP libraries must be installed on system [3], but more details could be found in [4]. Unfortunately the available support for SNMP in Asterisk is rudimentary, either as sub-agent for AgentX protocol, either as a standalone agent. We preferred the first case and thus within the configuration file `/etc/snmp/snmpd.conf` the AgentX support had to be enabled [5].

```
# Enable AgentX support
  master agentx
# Set permissions on AgentX socket
  agentXPerms 0660 0550 nobody asterisk
```

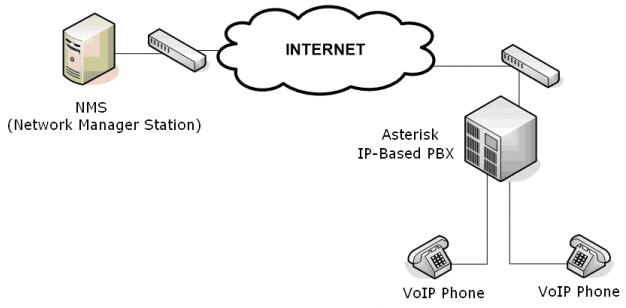

Figure 1. System architecture

Note the AgentX framework defines:

- a single processing entity called the master agent, which sends and receives SNMP protocol messages in an agent role but typically it has little or no direct access to

Manuscript received September 8, 2009, revised October 24, 2009

management information.

- none or more processing entities called subagents, which are "hidden" from the SNMP protocol messages processed by the master agent, but they have access to management information. Communication model used by AgentX protocol is presented in Figure 2.

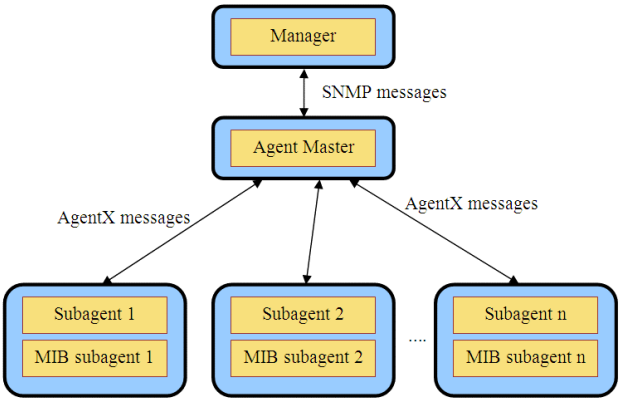

Figure 2. Communication model used by AgentX

The Agent Master and Subagent entities communicate via AgentX protocol messages, but this is invisible to manager entity. The advantage is related to the possibility to connect and to register several managed objects without having to interrupt the management service. This mechanism produces a more flexible management system, where each subagent can be closer to the managed information.

Both the manager and the agent are using UDP port 161 for communication, but the manager is waiting for notifications at UDP port 162. The exchange of information (except the alarms) is described in Figure 3.

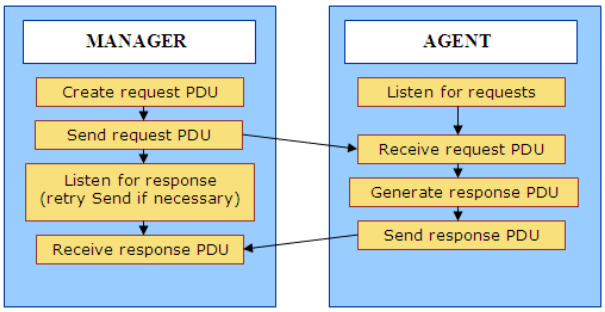

Figure 3. Communication between manager and agent

In order to obtain information about users' calls, we used .log files generated by Asterisk for every finished call. The type of information stored into this file is configured in `cdr_custom.conf` located in `/etc/asterisk`. Usually the CDRs are stored in a CSV (Comma Separated Value) file in `/var/log/asterisk/cdr-csv` folder. The default file is `Master.csv`, but it is possible to specify a different accountcode (with SetAccount or in `sip.conf`, `iax.conf` or `zaptel.conf`) and other files can be created.

CDRs can also be stored in a database. Asterisk currently supports SQLite, PostgreSQL,MySQL and unixODBC. Many people prefer to store their records in a database, so that queries can be run to help with billing and resource management. Our current approach was to use CSV files, and we let the option for database to be implemented later. The file `cdr_custom.conf` contains the following:

```
; Mappings for custom config file
;
[mappings]
Master.csv=> "${CDR(clid)}","${CDR(src)}",
"${CDR(dst)}","${CDR(dcontext)}",
"${CDR(channel)}","${CDR(dstchannel)}",
"${CDR(lastapp)}","${CDR(lastdata)}",
"${CDR(start)}","${CDR(answer)}",
"${CDR(end)}","${CDR(duration)}",
"${CDR(billsec)}","${CDR(disposition)}","${CDR(ama
flags)}","${CDR(accountcode)}","${CDR(uniqueid)}",
"${CDR(userfield)}"
```

The variables that are defined in the above file have the following interpretation:

`${CDR(clid)}` = callerid for the call
`${CDR(src)}` = callerid number for the call
`${CDR(dst)}` = destination extension
`${CDR(dcontext)}` = Destination context
`${CDR(channel)}` = Src channel
`${CDR(dstchannel)}` = Destination channel if appropriate
`${CDR(lastapp)}`= this is the last application in the dialplan used, on an outgoing call this will be DIAL.
`${CDR(lastdata)}` = these are the parameters given to the last application used in the dialplan
`${CDR(start)}` = time of the start of the call
`${CDR(answer)}` = time when the call was answered
`${CDR(end)}` = time when the call got hung up
`${CDR(duration)}` = duration of the call
`${CDR(billsec)}` = duration of the actual call (without the ringing)
`${CDR(disposition)}` = status of the call (ANSWERED, BUSY, NO ANSWER)
`${CDR(amaflags)}` = flag for the type of CDR
`${CDR(accountcode)}` = the accountcode as set for this channel
`${CDR(uniqueid)}` = a unique id for this call
`${CDR(userfield)}` = a userfield set by the dialplan command SetCDRUserfield

In order to obtain information, the *Master.csv* file is parsed by an application called *parserd,* specially developed for this purpose. After a specific amount of time it will read the file and update the MIB with new CDR information. The interval used by the program is configurable (from 5 seconds up to 24 hours) and is passed as input parameter to the program. By default this parameter is 30 minutes. Programming techniques used in *parserd* application follow specifications from [6].

```
[root@P168 SNMP]# ./parserd -t 30 -f SNMP_Data.xml
parserd -- Master.csv file parser
Usage : parserd [-t time] [-f filename]
        filename : name of the file where data will be stored
        time : time interval between parsing
AST_Parser: Parser  started at : Mon May  4 13:52:43 2009
AST_Parser: Parsing at : Mon May  4 13:53:13 2009
AST_Parser: Parsing at : Mon May  4 13:53:43 2009
AST_Parser: Parsing at : Mon May  4 13:54:13 2009
```

Figure 4. Launching the parserd program

Information extracted by this dedicated software is stored into an `.xml` file, which is used by SNMP agent for providing responses to manager queries. By default the file is `SNMP_Data.xml`. Objects defined in original MIB are obtained directly from Asterisk. The agent implements SNMPv1, v2c and v3, the last version being superior with respect to security features. Thus its Security Subsystem

handles the authentication and privacy of SNMP data, having support for both community string-based security (as in v1 and v2) and for specific user-based authentication.

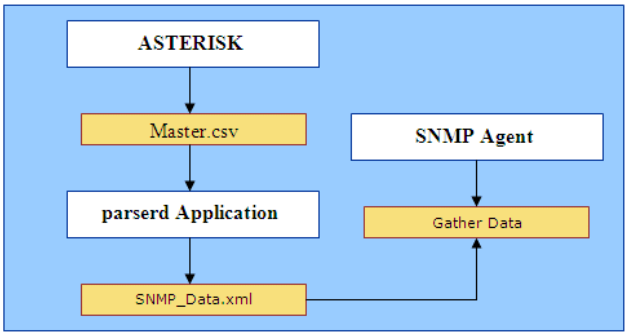

Figure 5. Communication between applications running on Asterisk machine

To be more concrete, the user-based authentication involves MD5 and SHA algorithms without sending a password in clear. Timeliness protects against message stream modification or delays by time-stamping each SNMP message. Finally, the privacy service uses the DES algorithm to encrypt and decrypt SNMP messages. Currently, DES is the only algorithm used, although others may be added in the future. Regardless the version, Figure 6 describes how SNMP requests are processed within the proposed agent.

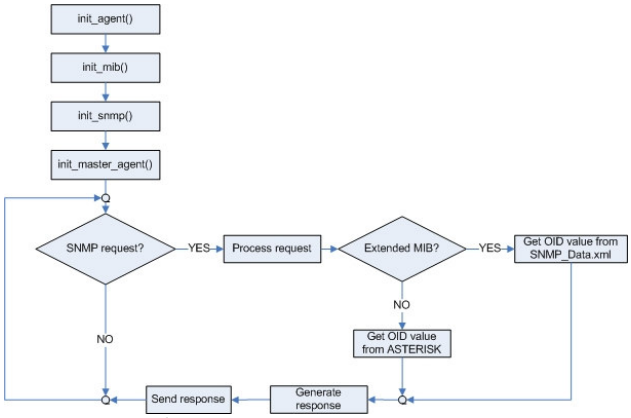

Figure 6.Processing requests in SNMP agent

III. MIB DETAILS

Management information is viewed as a collection of managed objects, residing in a virtual information store called MIB (Management Information Base). Collections of related objects are defined in MIB modules, written in a subset of ASN.1 [7] and following the guidelines specified in [8]. The original MIB included in Asterisk has five sections described in Table 1 and defined objects presented in Figure 7.

Table 1. Parameters of the original MIB

Section	Description
`asteriskVersion`	Information about version.
`asteriskConfiguration`	Information about processes.
`asteriskModules`	Number of modules loaded.
`asteriskIndications`	Indications used by Asterisk
`asteriskChannels`	Details about active channels

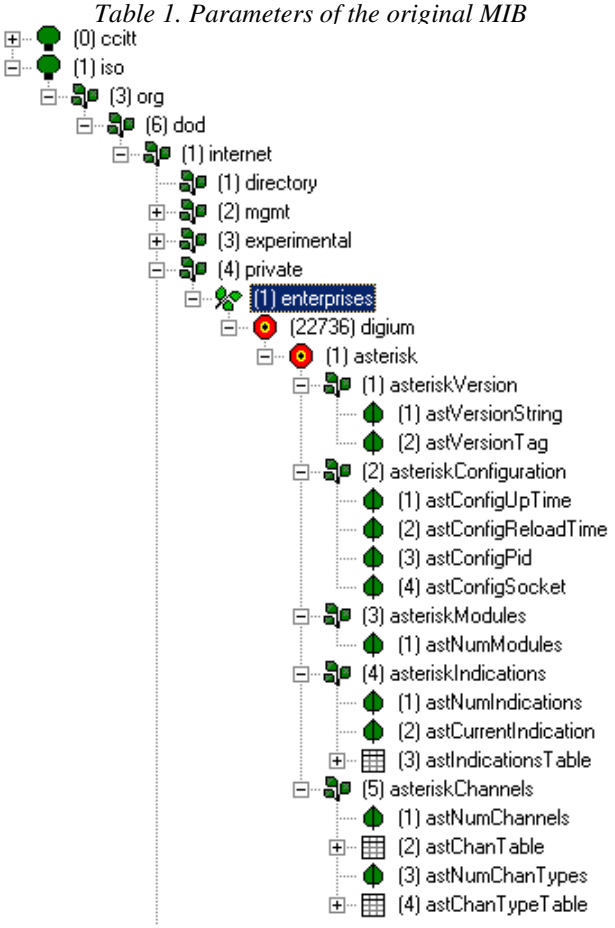

Figure 7.Defined objects in Asterisk MIB

Note that there is no information about calls made or received by clients managed by Asterisk. After analyzing the log files from PBX, we decided that new objects regarding the calls can be added to the MIB. Our proposal is to extend the sections already provided with a new one called `asteriskCallStatistics` (OID `1.3.6.1.4.1.22736.6.`), and to define 7 objects. The proposed MIB structure is described in Figure 8 and Table 2.

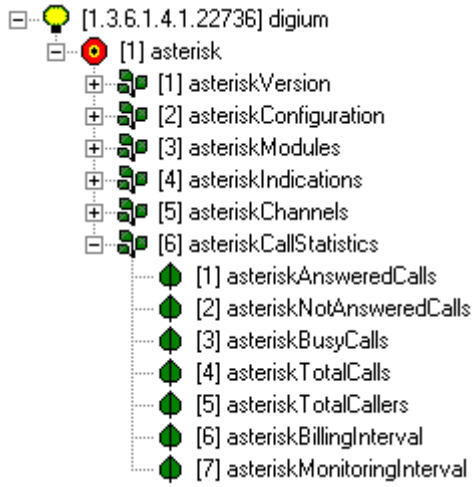

Figure 8.Proposed objects to be added in Asterisk MIB

Parameter/OID	Description	Value type
`asteriskAnsweredCalls` `1.3.6.1.4.1.22736.6.1`	Total number of calls initiated in Asterisk which were answered. The value *ANSWERED* of the field CDR (disposition) is counted on each row within *Master.csv* file.	Counter32
`asteriskNotAnsweredCalls` `1.3.6.1.4.1.22736.6.2`	Total number of calls initiated in Asterisk which were not answered. The value *NO ANSWER* of the field CDR (disposition) is counted on each row within *Master.csv* file.	Counter32
`asteriskBusyCalls` `1.3.6.1.4.1.22736.6.3`	Total number of calls initiated in Asterisk which were finished because of busy call. The value *BUSY* of the field CDR (disposition) is counted on each row within *Master.csv* file.	Counter32
`asteriskTotalCalls` `1.3.6.1.4.1.22736.6.4`	Total number of calls initiated in Asterisk. The value of this OID is the sum of values within previous fields.	Counter32
`asteriskTotalCallers` `1.3.6.1.4.1.22736.6.5`	Total number of different callers with monitoring interval.	Counter32
`asteriskBillingInterval` `1.3.6.1.4.1.22736.6.6`	Total number of billing seconds. This is obtained as the sum of values in the field CDR (billsec) on each row within *Master.csv* file.	Counter32
`asteriskMonitoringInterval` `1.3.6.1.4.1.22736.6.7`	Monitoring interval. It contains in string format StartDate of the monitoring and the number of secons until the end of the monitoring interval.	Octet String

Table 2. Parameters of the extended MIB

Note that we selected to put the MIB under enterprises branch, using 22736 as starting point, because this OID was allocated to Digium.

IV. EXPERIMENTAL RESULTS

All experiments were conducted using Asterisk 1.4.7 running under Linux Fedora Core 8. The agent used in experiments is based on Net-SNMP libraries and could get services at UDP port 161. Eventually the port number could be changed.

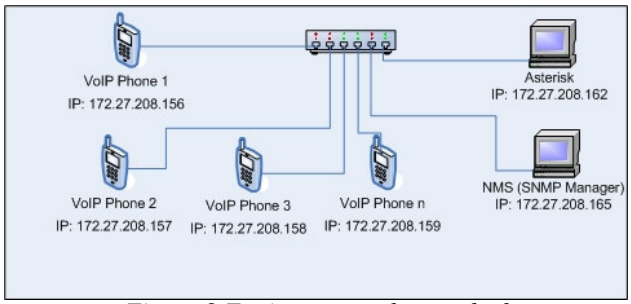

Figure 9.Equipment used as testbed

Several VoIP SIP phones initiated calls through Asterisk, the statistics about traffic being recorded in *Master.csv* file stored on a separate PC running both Asterisk and SNMP agent. Figure 10 represents a "snip" of *Master.csv* resulted.

```
8001" <8001>","8001","12","internal","SIP/8001-083d9538","SCCP/12-0000000b","VoiceMail","u12@defaul
P7970-B2" <12>","12","0741160376","internal","SIP/12-0928bd50","Zap/1-1","Dial","Zap/1/0+0741160376",
P7970-B2" <12>","12","0744267970","internal","SIP/12-09283c50","Zap/1-1","Dial","Zap/1/0+0744267970",
P7970-B2" <12>","12","8001","internal","SIP/12-09283c50","SIP/8001-0928a7d0","Dial","SIP/8001|20","200
P7970-B2" <12>","12","8001","internal","SIP/12-09283c50","SIP/8001-09289250","Dial","SIP/8001|20","200
8001" <8001>","8001","12","internal","SIP/8001-09289250","","VoiceMail","u12@default","2009-02-02 19:23
```

Figure 10. Content of Master.csv

Parserd is running on the same PC as Asterisk and it was configured to process data from *Master.csv* every 10 minutes. The results are presented Figure 11, where the *SNMP_Data.xml* file may contain around 5,500 records taken in less than one second.

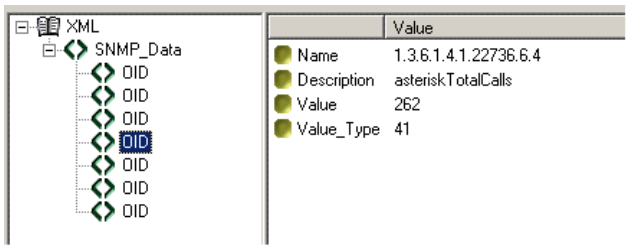

Figure 11. Content of SNMP_Data.xml

The algorithm implemented in *parserd* is discussed in Figure 12. Note that the management console used was an application called *SNMP Manager* developed also by our team. It will perform polling of SNMP agent to obtain information about calls made through Asterisk.

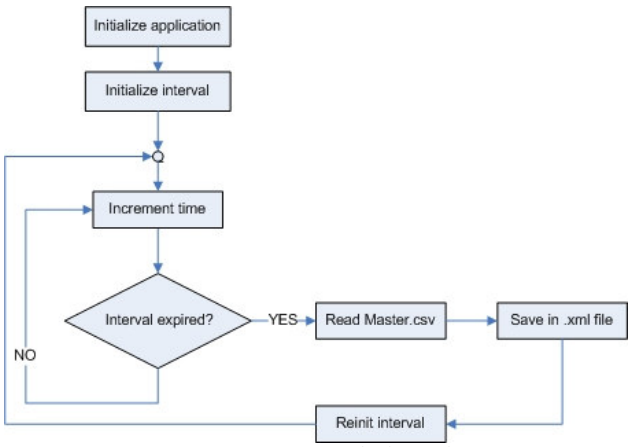

Figure 12. Parserd algorithm

```
E:\net_snmp>snmpget -v 1 -c public 192.168.2.160 1.3.6.1.4.1.22736.6.4
SNMPv2-SMI::enterprises.22736.6.4 = INTEGER: 262

E:\net_snmp>_
```

Figure 13. SNMPv1 agent on Asterisk using net-snmp

```
E:\net_snmp>snmpget -v 3 -l noAuthNoPriv -E 8000000201C0A802A0 192.168.2.160 1.3.6.1.4.1.22736.6.4
SNMPv2-SMI::enterprises.22736.6.4 = INTEGER: 262

E:\net_snmp>
```

Figure 14. SNMPv3 agent on Asterisk using net-snmp

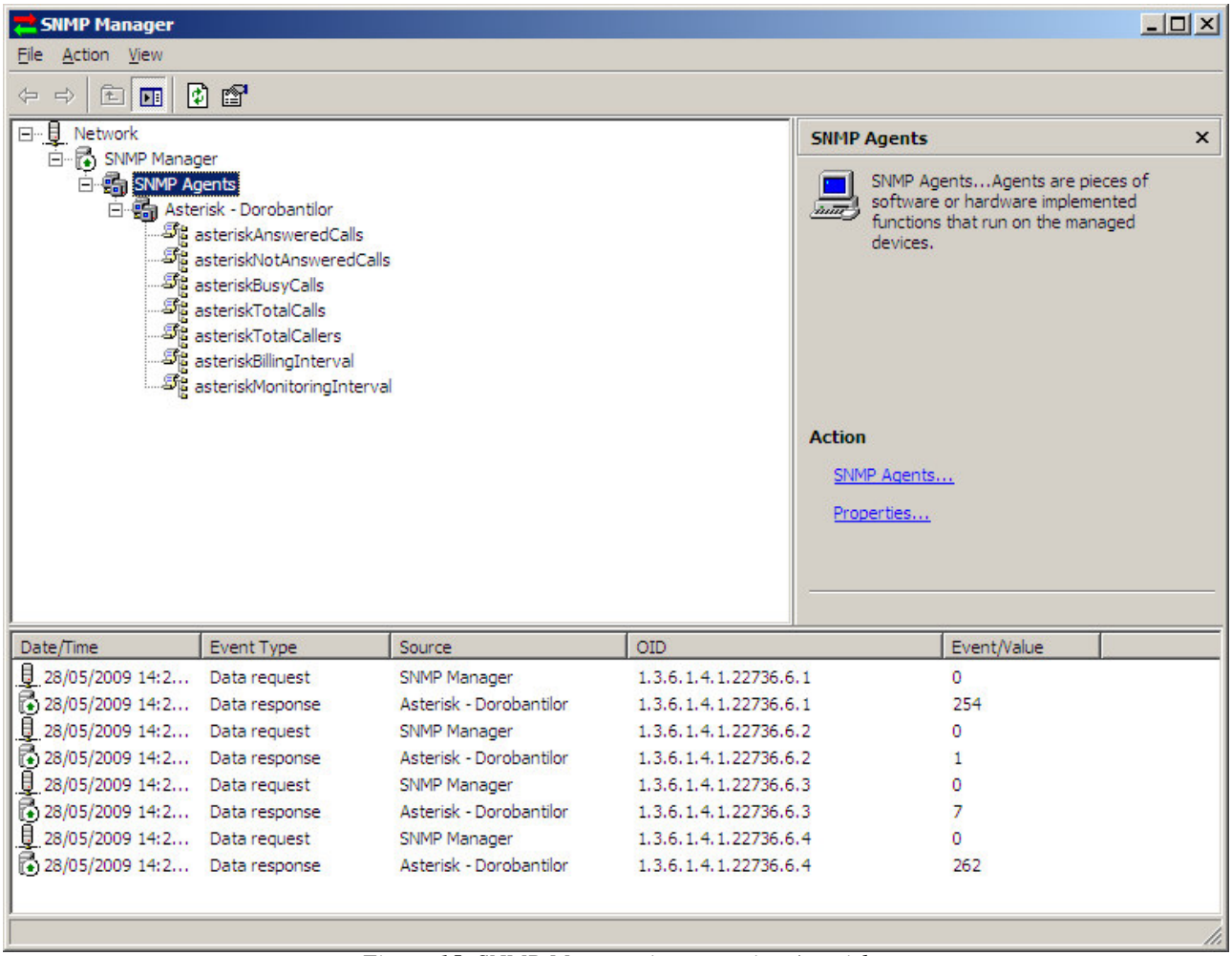

Figure 15. SNMP Manager interrogating Asterisk

V. CONCLUSION AND FUTURE WORK

The contribution does not refer to the management of new access technologies, being focused on MIB extensions for VoIP applications. Seven objects characterized by length and value have been defined to offer relevant information about total number of calls and other statistics about the traffic (with answer, no answer, busy call, billing seconds, monitoring interval). Although the management information is devoted to a concrete situation (calls in Asterisk), the idea was in fact to demonstrate the feasibility of integrating applications and networks management using SNMP, a generalized non-proprietary protocol.

Updates folowing the moment we wrote this paper may include Asterisk 1.4.24.1 (stable version) and 1.6.0.9 (beta version) running Linux Fedora Core 11.

Once the needs for databases will arise, SQLite, PostgreSQL or MySQL may be employed. The current version of SNMP Manager does not have a module for automatic interpreting of the information regarding calls. A module could be added to generate statistics, alarms, graphical representations etc.

REFERENCES

[1] W.Stallings, *SNMP, SNMPv2, SNMPv3, and RMON1 and 2, Third Edition*, Addison Wesley, New York, 1998.
[2] J. VanMeggelen, J.Smith & L.Madsen, "*Asterisk™ The Future of Telephony. Second Edition"*,O'Reilly Media Inc, 2007
[3] ***, Net-SNMP Manual Pages, *SourceForge Net Project*, 2009, http://www.net-snmp.org/docs/man/.
[4] ***, HowTo: Monitor Asterisk with SNMP, *Voxilla* 2009,

http://voxilla.com/2009/02/03/configuring-asterisk-snmp-support-1131
[5] M.Daniele, B.Wijnen, M.Ellison, D.Francisco, „Agent Extensibility (AgentX) Protocol Version 1", *RFC 2741*, IETF, January 2000
[6] W.R.Stevens, S.A.Rago, *Advanced Programming in the UNIX(R) Environment. Second Edition*, Addison-Wesley Professional, 2005.
[7] ***, „Information Processing Systems - Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)", *International Standard 8824,* ISO 1987.
[8] K.McCloghrie, D.Perkins, J.Schoenwaelder, "Structure of Management Information Version 2 (SMIv2)", *RFC2578,* IETF April 1999.