NeMoS: NETWORK MONITORING WITH SOUND

Delfina Malandrino, Daniela Mea, Alberto Negro, Giuseppina Palmieri, Vittorio Scarano

Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Università di Salerno
84081, Baronissi (Salerno), Italy
{delmal,alberto,palmieri,vitsca}@dia.unisa.it

ABSTRACT

In this paper we present NeMoS, a program written in Java that allows monitoring of a distributed system with sound. The architecture is client/server: the server collects (by polling via SNMP [16]) data from the monitored Network Components and the client plays accordingly. The sonification technique associates events (as defined by the user) to MIDI tracks.

Our system is versatile (several channels of events can be created and used), easily configurable (personalization of events and tracks is offered to users), standard (it fits within the framework described in RFC 2570 [5]), distributed (multiple clients can be anywhere in the system) and portable (using Java as Programming Language).

1. INTRODUCTION

Distributed computer systems are nowadays the most widespread architecture in computer science. Efficient and reliable services are given to users by leveraging on the inherent fault-tolerance and redundancy that is provided by the distributed hardware and software architecture of the system. It is, therefore, crucial to offer timely information about the status of the services. The task is particularly complex, at least as much complex as the heterogeneous nature of all the services in the network: e-mail, web servers, file sharing, CPU (over different machines), printers, routing, naming services (DNS), etc.

The monitoring and management of distributed systems is well recognized as a key issue in the industry: many resources are devoted in a company to make the system work "around the clock". Several important and complex solutions are provided by major software companies, such as Computer Associates (Unicenter) [15], HP (OpenView) [19] and IBM (Tivoli) [9].

In this paper we describe a prototype, NeMoS, that is able to monitor the behavior of distributed systems by representing with the sound the information provided by the Simple Network Management Protocol, a well-known and widespread standard for monitoring devices on the network.

We argue in this paper that it is really important to complement visual feedback to the system manager by audio information: sonification of the system provides an effective, efficient and unobtrusive way to monitor in *real-time* the behavior of the system. Very often, the attitude of any manager is "No news, good news" i.e. if nothing is signalled to the manager[1] it means that everything is ok. In other words, the management is *problem-centered* and does not provide information of the ongoing activities of the system. This kind of information can be critical in anticipating problems so that prompt action can be taken by the system manager to anticipate malfunctioning and/or fine-tune the system.

Sonification, i.e. the usage of non-speech audio to convey information, is particularly useful when there is an abundance of data to be considered. While sonification of data is not new (the "ticking" of a Geiger counter as well as sonars are well known), sonification is a technique used successfully in several contexts. For example, sonification can be used to allow blind scientists to examine experimental data via an auditory presentation (as for infrared spectrographic data [14]).

More relevant to our field is another application of sonification, i.e. the usage of sound to monitor the behavior of a computer system. As a matter of fact, numerous anecdotes report several examples of very early monitoring by sonification. A first example can be the monitoring of a garbage collector in an early prototype computer Olivetti 9104 at the IAC (*Istituto per le Applicazioni del Calcolo* of the Italian Council of Research (CNR)): sounds with different tones indicated the memory regions where the garbage collector was working [1]. As another example of early usage we can cite the first business oriented computer, named Lyons Electronic Office (LEO) [11]. This machine had a speaker that allowed the programmers (used to certain frequency variations) to monitor programs and detect if something was wrong.

Entirely devoted to network monitoring, SoundWIRE [6] is a utility for measuring the reliability of bidirectional network connections by the sonification of an enhanced "ping" between a pair of hosts. The internet connection itself is converted into a vibrating acoustic medium, producing a resonating tone which can be understood according to musical criteria: for example, the pitch shows the roundtrip time (RTT) while stability of pitch indicates regularity of service and lost packets create pops and distortions. A listener can easily detect even very subtle changes to QoS.

As another successful usage of sonification for monitoring a rather complex environment, we can cite ShareMon system [7] that conveys status information of a server in the background using subtle metaphorical sounds (such as a knocking sound as a login on the server).

As last example of previous work in this area, we briefly describe Peep [10]. It is the only work (that we are aware of) that monitors complex systems, dealing with multiple and heterogenous data sources. Particularly interesting is their musical choice: they provide a mapping of events to "natural" sounds (like birds, crickets, etc.). As we will say later, we agree with their choice of an abstract "musical" representation of data since it makes the usage of auditory peripheral information possible for a long time. Their architecture is based on clients (daemons that collect infor-

[1] using automatic systems, such as e-mail, SMS, pager or simply by angry phone call by users!

mation) and servers (that play the sounds). We compare NeMoS and Peep in Section 5.

2. NeMoS: REQUIREMENTS AND DESIGN

The design of NeMoS was based on the main requirement to provide peripheral, on-line and configurable information about the state of several components of a (possibly) large distributed system by using the sound. Current systems, in fact, offer real-time monitoring of a network but they limit themselves to page people when something seems to be going awry. NeMoS' characteristics reverts the approach: sound is used as peripheral information and provides a musical background to everyday activities so that the system manager can listen to the monitored components.

Another important characteristics of NeMoS' sonified interface is that it allows monitoring different parts of the same network system at once: the information are merged into a single musical flow thus representing the whole state of the part of the system the system manager is interested in. At the same time, it is easy to shift the focus of our system to different (previously defined) aspects that require attention by the system manager.

In fact, the system manager can configure different channels consisting of a number of events to be monitored for a specified Network Component. Each event is associated with a track (or multiple tracks) of a MIDI file chosen (and configurable) by the user for each channel. NeMoS can monitor two different types of events with two different musical representation: simple (*digital*) events such as a printer in "No paper" status, or a measurement being above a predetermined threshold; complex (or *analog*) events that correspond to the increasing value of a measurement whose representation is a set of MIDI tracks that are played (according to the configuration) as the value increases.

3. NeMoS ARCHITECTURE

3.1. A standard framework

NeMoS follows the "Architecture of the Internet Standard Management Framework", as described in RFC 2570 [5]. The main components of this architecture are: Network Components (i.e. the monitored network element), Agents, Network Management Stations (NMSs), Management Information Bases (MIBs), and finally the Network Management Protocol (SNMP) that allows the transmission of the managed information between Agents and NMSs.

On each managed Network Component there is a software module (called Agent) that collects and stores local management information. The NMSs are usually workstations that show graphic information about the monitored events. On each NMS there is a software module, called Manager, that deals with the communication with the Agents on the managed Network Components. Information is collected either actively by the Manager, through the polling of the Agents, or passively, through *traps* sent by the Agents. The communication between Agent and Manager usually occurs via SNMP.

The objects that contain management information are stored, by each Agent, in a special database called Management Information Base (MIB): the MIB of a router, for example, maintains information about its routing table, the MIB of a printer maintains information related to its state (low toner, no paper, printing, paper jammed, etc.). Several RFCs[2] describe the standard MIB for Network Components (i.e. routers) [13], hosts [8] and printers [18].

Any MIB is a collection of objects which define the properties of the managed object within the device to be managed. The set of defined object has a tree structure. The Object IDentifier (OID) of an object in the MIB is the sequence of non-negative Integer value traversing the tree to the node required. Then, to read the value of objects in a MIB one must *browse* (navigate) the MIB tree structure until reaches the desired identifier on a leaf.

Many are the commercially available Network Monitoring Systems based on this framework, such as ActiveSnmp [17] and HP OpenView, but they are strongly based on a visual interface.

In this context, NeMoS is a Network Monitoring System that uses a audio interface rather than a visual one. NeMoS uses SNMP as Management Protocol that is, at the moment, the most popular Management Protocol on Internet. SNMP is an application-layer protocol designed to facilitate the exchange of management information between network devices. By using SNMP, system managers can more easily manage network performance, find and solve problems, and plan for a smooth system growth.

3.2. NeMoS' Overview

NeMoS' client-server architecture consists of two components: NeMoSServer and NeMoSClient. The NeMoSServer manages the communications with the Agent of the monitored Network Components to obtain the management information and then send them to the NeMoSClient. The NeMoSClient analyzes the information received and produces the sound. It also provides a simple user interface to show the monitored Network Components and browse their MIBs.

A client plays accordingly to events that occurs. An *event* consists of: the hostname (Fully Qualified Host Name or IP address) of the Network Component, a variable to monitor (from a specific object of the MIB), an (optional) threshold (or value), some (optional) parameters and a MIDI track associated. Depending on its nature, the event is triggered (and the corresponding MIDI track played[3]) if the variable has reached a given value (or passed a given threshold). As an example, one can monitor if the CPU load of a given machine (`hrSWRunPerfCpu`) is over 20% by using the appropriate variable in the MIB and setting the threshold to 20. As another example, one can define the event as a value of the variable `tcpConnState` that represents the state of the TCP connection on a specific network interface (as first parameter) and with a specific TCP port (as second parameter): for example the value 2 indicates that the connection is in *Listen* state.

We have also included analog events by grouping several identical MIDI tracks to an event where we define a starting value v (when reached, the first MIDI track is played) and a step value s: every time the variable reaches value $v + i \cdot s$ then the first $i + 1$ MIDI tracks are played together. The effect is as acting on the volume by having several tracks playing at once.

By using the GUI of the NeMoSClient, the system manager can create (and later modify or delete) several channels. A *channel* is built by putting together several events and associating an appropriate MIDI file. To facilitate channels' composition, we have grouped the events that can be monitored in categories according to the nature of the Network Component and/or the nature of the information to monitor. There are seven categories: Security, Network, Host Resources, Processes, TCP and UDP Services, Device

[2] In the rest of the paper we will refer to standard identifier for MIB objects referred in these RFCs by using `typewriter` font.

[3] The track is played until new information reaches the client, i.e. every second.

Status and Printers.

We also provide some events whose value is evaluated by using different MIB variables: for example the percentage of "ICMP Echo Request" message received is evaluated with respect to the total number of ICMP messages received, i.e. (`icmpInEchos` over the total number of packets `icmpInMsgs`).

The system manager can compose several channels that are saved in a configuration file (so that they are automatically reloaded at startup by the NeMoSClient), modify them, and choose which one to play and easily switch among channels.

3.3. Architecture Components

NeMoS architecture is client-server and here we describe, with more details, its components. They are all written in Java and therefore our application is extremely portable.

NeMoSServer consists of two modules: the *NetManager* module and the *Trapd* module. The former performs the network discovery at start-up, and manages all connections with the clients; the latter waits for traps generated by the Agents. When a NeMoSClient issues a request connection, the *NetManager* checks, first, if the client is authorized, and then, if it is the case, initiates the connections. The *Trapd* module in the NeMoSServer receives the traps from Agents and sends the proper information to each client that has established a connection with the NeMoSServer. The client will show the trap by a visual message with a loud beep[4].

By using a parameter (i.e. `-nd`) on the command line, the NeMoSServer initiates a network discovery whose parameters (max number of hosts, max depth, etc.) are stored in a configuration file[5] `nms.conf`. Once finished the (potentially long) network discovery, all the information on the discovered Network Components are stored in a file `netMap.conf` so that it will be available next time the NeMoSServer is started without the parameter `-nd`.

NeMoSClient main goal is receive data from the server and play accordingly. It also provides a GUI to create, modify or delete channels. Moreover the client provides a GUI that allows the user to browse MIBs of the Network Components.

The NeMoSClient consists of two modules, the *NetBrowser* and the *EventsPlayer*. The first module allows the MIB browsing on a Network Component while the second, the *EventsPlayer*, sends to the *NetManager* the channel chosen by the user. Then the *NetManager* performs the continuous polling of the MIB for the value of the objects and regularly sends back the results to the *EventsPlayer*.

The *EventsPlayer* module is also in charge to play the sounds related to the monitored events by using the Java Sound API. When the user chooses the channel to monitor, the *EventsPlayer* instantiates a new module, the *EventsCollector*, that deals with receiving and storing the results in a format that facilitates the computation of the statistics by the *StatisticsMaker* module that communicates back to the *EventsPlayer* (each second) the MIDI tracks to activate.

[4]SNMP traps (included in our system only for completeness) are really rare and serious events (such as shutdown/restart of a computer) and therefore do not fit well in NeMoS objectives of background monitoring of a live system by musical representation.

[5]Other configuration parameters, such as the IP addresses of authorized NeMoSClients, are stored in this file.

4. CASE STUDY

Here we report our experience on using NeMoS in our Department. We first describe the criteria for the choice of the MIDI tracks and later describe some example configurations of the channels.

4.1. Peripheral Information through Music

Monitoring distributed systems is usually a task that is performed in the background of other activities. NeMoS offers to system managers information delivered by sound in such a way that they can "feel" the behavior and promptly recognize anomalies. As a matter of fact, several studies [2, 7] recognize the importance of peripheral information (i.e., information not critical but potentially useful nevertheless [12]) when it is delivered through sound, since sound can be effective in conveying additional information [3] while not taking precious screen space.

As a consequence, we decided to use NeMoS with MIDI files that can effectively generate "background music". In our previous experience in the sonification of a Web Server [3, 4] we found a useful combination of MIDI tracks that represented a musical structure that is neutral with respect to the usual and conventional musical themes. In fact, the goal was to make possible to hear the music generated for a long time, without inducing mental and musical fatigue: well-identifiable musical patterns repeated for a long time can provoke the psychological rejection of the information provided by our tool. As a consequence, we tried to avoid to configure harmonic-tonal fields as well as rhythmic references that are potentially able to attract the focus of the users by leveraging on their mnemonic and musical (personal) capabilities. Therefore we leave to timbre and duration the task to represent the information making usage of neutral and somewhat unusual MIDI timbres.

4.2. Some example channels

We describe the configuration of some channels that we have used to monitor the distributed system in our Department. The core of our system is mainly based on 3 servers: 2 applications and file servers (Linux and Windows Terminal Server) where users[6] log in and run their applications and a Digital Unix server for services as mail, DNS, WWW, FTP, etc. There are several workstations/PCs (around 40-50), three network laser printers (one is a color laser printer) and a router for the connection to the rest of our University network. The monitoring required must address traditional issues such as early detection of malfunctioning and preventing possible problems. In the first category we found useful to monitor printers malfunctioning (such as paper jammed, no toner, no paper), disconnection of the network with the University network and servers' problems (such as low disk space for file servers). In the second category, we were interested in preventing malfunctioning (i.e. printers with low toner or low paper, or unusual CPU/memory load on servers) as well as preventing possible malicious attacks such as "denial of service" attacks by IP flooding (that, for example, can be recognized by monitoring the router).

The MIDI file that we used was designed according to the consideration in the previous subsection. The Tracks 1 to 8 uses timbre *BowedGlass* (timbre 93 of the MIDI standard), track 9 uses *Woodblock* (timbre 116), track 10 *MusicBox* (timbre 11) and track 11 a very recognizable *Timpani* (timbre 48). The idea is that tracks 1 to 8 can be used either for analog events (expressing the increasing

[6]Users are researchers, faculties and PhD students of the Department.

load, for example) or to monitor a combination (a logical OR) of digital events (any printer without paper, for example). Tracks 9, 10 and 11 are used for digital events that must be easily recognizable.

We configured and used several example channels. We report here, for brevity, only some of them i.e. the *Printers* channel and *System load* channel. The *Printers* channel monitors by tracks 1 to 6, respectively, the three network printers for the two conditions "Low Paper" (`hrPrinterDetectedErrorState` with the 7^{th} bit set) and "Low Toner" (`hrPrinterDetectedErrorState` with the 6^{th} bit set). On tracks 9, 10 and 11 we monitor the "Printing" state (`hrPrinterStatus` equals to 4), mapping track 11 (the recognizable louder *Timpani*) on the laser color printer, whose usage should be consistently less frequent (given the cost per page).

The *System load* channel monitors the load of the 2 application servers, respectively, on tracks 1 to 4 and 5 to 8. On track 9 we monitor when the percentage of used file on the file server goes above a threshold (in our example, 50%). On track 10 we monitor the percentage of "ICMP Echo Request" (`icmpInEchos` over the total number of packets `icmpInMsgs`) received by the host (to recognize flooding denial-of-service attacks) and, finally, on track 11 we placed the (critical) event "Interface is Down" (`ifOperStatus`) on the router.

Our experience of using NeMoS is that a certain tuning is necessary, especially when thresholds are to be determined: for example what is a "normal" level of CPU load during the day, what is the "usual" disks occupancy, etc. In this context, precious is the configurability and personalization offered by NeMoS as well as the possibility to configure (and easily switch among) many different channels.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented NeMoS, its architecture and application to a real case. The sonification technique for monitoring complex systems seems to be a promising complementary tool to the usual visual interface.

While several applications of sonification were developed to monitor specific services [6, 7, 3] our goal is to develop a framework to include sound as a possible interface to monitor **all** the components of a network that can be usually monitored via the standard Internet Standard Management Framework (RFC 2570 [5]) and its communication protocol SNMP.

The only previous work that had the same all-encompassing objective as ours is Peep [10]. The main substantial differences between Peep and NeMoS lie in our usage of a standard protocol such as SNMP. This characteristics, in fact, allows NeMoS to monitor for devices (such as network printers or routers) that cannot be monitored by Peep, since its "monitors" are to be executed as daemons on the machine and, therefore, require both programmability of the device[7] and accessibility to the machine (you must own enough privilege to run programs on the node). As a second consequence, NeMoS does not require programming (at least in the current configuration, see later in this section) and is easily configurable/personalizable. Another consequence is the fine-tuning of the parameters that can be monitored: an example can be the ability to monitor (provided by SNMP) when somebody is putting paper on the printer and when the toner is low.

Future work on NeMoS include several directions: first of all, we are planning to include the MIBs for explicitly monitoring mail servers and WWW servers (by using RFCs 2789 and 2594). Then, we would probably work toward offering an environment to independently develop plug-ins (i.e. adding new MIBs) for NeMoS. Finally, an interesting step would be using an applet as a client in order to allow sonified remote monitoring via a universally available interface such as a web browser.

[7] You cannot execute daemons on a printer and the trick of monitoring printer queues on a host does not monitor the effective usage of a network printer (i.e. with multiple machines using it).

6. REFERENCES

[1] Giorgio Ausiello, Private communication as reported by Alberto Marchetti Spaccamela.

[2] Buxton, B. (1989). Introduction to this special issue on nonspeech audio. Human-Computer Interaction, 4, 1-9.

[3] M. Barra, T. Cillo, A. De Santis, U. Ferraro Petrillo, A. Negro, V. Scarano. "Personal WebMelody: Customized Sonification of Web Servers". Proceedings of the International Conference on Auditory Display (ICAD), 29 July - 1 August 2001, Espo, Finland.

[4] M. Barra, T. Cillo, A. De Santis, U. Ferraro Petrillo, A. Negro, V. Scarano. "WebMelody: Sonification of Web Servers". Poster Proceedings of the 9th Int. World Wide Web Conference(WWW9), May 15-19 2000, Amsterdam(Netherland).

[5] J. Case, R. Mundy, D. Partain, B. Stewart. "Introduction to Version 3 of the Internet-standard Network Management Framework", April 1999. RFC 2570.

[6] C. Chafe, R. Leistikow. "Levels of Temporal Resolution in Sonification of Network Performance". Proceedings of the 2001 International Conference on Auditory Display, July 29 - August 1 2001 Espoo, Finland.

[7] Cohen, J. "Monitoring background activities." In G. Kramer (Ed.), Auditory display: Sonification, audification, and auditory interfaces. Reading, MA: Addison-Wesley, 1994.

[8] P. Grillo, S. Waldbusser. "Host Resources MIB", September 1993. RFC 1514.

[9] "IBM Tivoli Monitoring for Network Performance" http://www.tivoli.com

[10] M.Gilfix, A.Couch. "Peep (The Network Auralizer): Monitoring Your Network with Sound". Proc. of 14th System Administration Conference (LISA XIV). Dec. 3-8, 2000, New Orleans (LA) USA.

[11] "Lyons Electronic Office". http://is.lse.ac.uk/Leo/HistoryCD_LEO.htm

[12] Maglio, P. P. Campbell, C. S. (2000) "Tradeoffs in the display of peripheral information", in Proceedings of the Conference on Human Factors in Computing Systems (CHI 2000).

[13] K. McCloghrie, M. Rose. "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II", March 1991. RFC 1213.

[14] Lunney, D., Morrison, R.. "High technology laboratory aids for visually handicapped chemistry students". Journal of Chemistry Education 58, 228 (1990).

[15] "Network and Systems Management" http://www3.ca.com/Solutions/

[16] "Simple Network Management Protocol". http://www.snmp.org

[17] "SNMP network management application" http://www.cscare.com/ActiveSNMP/

[18] R. Smith, F. Wright, T. Hastings, S. Zilles J. Gyllenskog. "Printer MIB", March 1995. RFC 1759.

[19] "Software Management HP openView". http://www.openView.hp.com