Performance Monitoring of Management Systems [*]

Wagner Meira Jr.
José Marcos Nogueira
Departamento de Ciência da Computação
UFMG

Patrícia Aguiar
Christiano Mata Machado
RPR
Telemar Minas

Belo Horizonte – MG 31270-010

`(meira, pati, jmarcos)@dcc.ufmg.br, christiano@telemig.com.br`

Abstract

Telecommunications Management Networks, when deployed in plants of medium or high complexity, can have highly complex structured and a great number of hardware and software components. In such cases, they are distributed systems with many interactions between the computational processes, where the correct behavior and good performance become critical issues in the operation of the telecom plants.

In this paper we present SisMonit, the performance monitoring system of the SIS - Integrated Supervision System. It is characterized by high flexibility and multiple functionalities concerning data acquisitions granularity and detail level of performance information generated.

1 Introduction

Telecommunications Management Networks (TMN), when deployed in plants of medium or high complexity, can have highly complex structured and a great number of hardware and software components. In such cases, they are distributed systems with many interactions between the computacional processes, where the correct behavior and good performance become critical issues in the operation of the telecom plants. As a result of its size and complexity, both the systems and management networks demand performance and fault management for themselves.

Performance monitoring of these management systems is necessary not only to supervise its normal tasks, verifying whether the computational costs are within expected computational costs, but also subsidize the evaluation of the system design, its configuration parameters, and possible needed changes. The performance tuning process of management systems, similarly to other parallel and distributed systems [3], is cyclic and each cycle can be divided into five phases that can be summarized by the following questions: (1) how well the system is performing; (2) if performance is not good, what kind of performance problem is affecting the system; (3) where the problem is ocurring; (4)

[*]This work was supported by Telemar Minas in a joint project with the Federal University of Minas Gerais. Telemar Minas is a Telecom operator of the Minas Gerais state/Brazil and is part of the Telemar System.

why the problem is ocurring; and (5) how we can solve the problem. A monitoring system must provide performance data that support determining answers for the first three questions. Obviously, automating all these phases is beyond the scope of a monitoring system, mainly the last two, which are still being targeted by current research [2].

There are several scenarios where performance management is essential, such as when a data capturing agent is overloading the machine processor, a database query for generating an alarm report is prohibitively expensive in terms of computation, or the latency for an operator-requested action is high. Moreover, in order to correlate performance measures to system activities, it becomes necessary to monitor internal events from the various modules, as well as their temporal and spatial relationships.

There are some management system characteristics that drive the strategies for implementing the performance monitor. The variety of entities and events that must be supervised demand flexible mechanisms in terms of acquisition frequency and granularity, which also aim to maximizing the amount of information about the system behavior with the smallest overhead possible. Management systems such as the Integrated Supervision System (SIS) [1, 6, 4] are composed by several modules, where changes are neither desirable nor easy to be performed. As a consequence, module's instrumentation can not be modified very frequently. Finally, since these systems are under continuous development, modules are frequently rebuilt, when functionalities are added, deleted or changed, demanding a monitoring system that is both adaptive and modular, so that monitoring a module do not disturb on the execution of other modules.

In this paper we present SisMonit, which is the performance monitoring system of SIS, a telecommunication management system developed by Telemar–MG and DCC–UFMG [1]. The next section describes SisMonit architecture and implementation. Section 4 describes how to employ the monitoring system. We illustrate the use of our monitoring system by describing how we monitor agents and managers in Section 5. The last section presents our conclusions and some future work.

2 Performance Monitoring of TMN Systems

Functionally, SIS is a three-level hierarchy of central units (Figure 1), a structure adopted by several telecommunications companies for their operations and maintenance centers.

The first level comprises UCPs (main central units) that are responsible for the management of the whole plant. The second level is composed by UCRs (regional central units) that supervise a region, allowing its autonomous operation. The lowest level is composed by UCSs (sub-regional central units) that supervises a sub-region, also allowing the sub-regions to work autonomously. UCSs are also responsible for gathering the information that is used by SIS, through interaction with supervision sub-systems (SSS), network elements themselves, or communication adapters.

One of the main characteristics of SIS's software architecture (Figure 2) is to have just one source code that is configured and installed to perform supervision tasks in any of the hierarchy levels. We can divide this software architecture into four major families of software modules: (1) system management; (2) access (managers, monitors, and agents); (3) DBMS interfacing, and (4) man-machine interfacing. Each central unit is implemented through several modules that exchange messages while performing supervising tasks. Notice that there may be several instances of managers, monitors, and interfaces to both DBMS and operators.

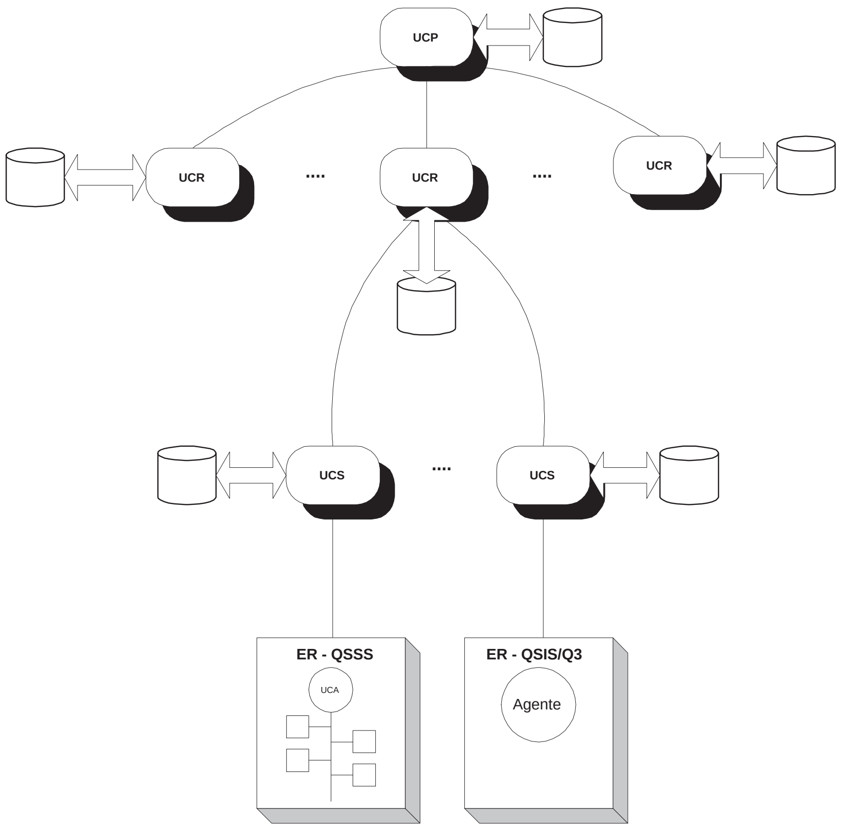

Figure 1: Functional Architecture

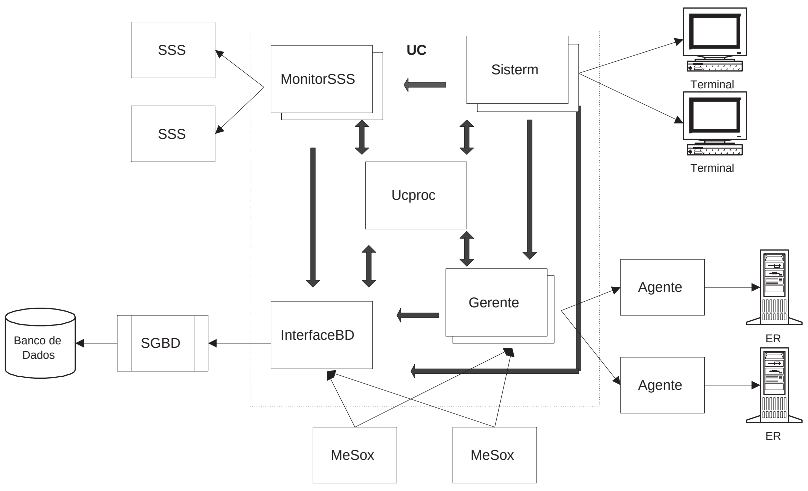

Figure 2: Software Architecture

For such a large system, performance monitoring is absolutely necessary for understanding performance phenomena such as requests presenting high turn-around times, large communication latencies between modules, and contention for the database system. Thus, from this brief description, we can identify some of the system's characteristics that are relevant for the sake of performance monitoring:

- the system is completely distributed and its components are heterogeneous, from both software and hardware perspectives;
- the connectivity among modules also varies significantly, ranging from Ethernet to slow twisted pair links;
- the system is being enhanced continuously, as a consequence of technological advances and new services to customers; and
- modules are inserted, removed or modified all the time, and changes should be restricted as much as possible to a single module.

Given the number, dynamicity, complexity, and diversity of SIS's components, there are some requirements that must be fullfilled for a performance monitoring system:

Location capability: performance data should contain location information that allows users to quantify the performance of not only the various modules but also internal functions and events.

Functionality Correlation: it should be possible to correlate activities associated with the same events and tasks in different modules.

Non-intrusive: The monitoring task should not disturb the performance of the monitored modules.

Flexible: the monitoring strategy may change during the module execution without affecting the module functioning.

In the next section we describe SisMonit, a monitoring system we developed that satisfies all these requirements.

3 SisMonit: Design and Implementation

3.1 Monitoring Strategy

Since the target applications are very diverse as well as their monitoring requirements, the monitoring strategy should emphasize flexibility and simplicity without reducing the quality and the level of detail of the monitored data. We define our monitoring strategy from four perspectives: storage, acquisition, monitoring level, and notification. These perspectives are discussed in the sections that follow.

3.1.1 Storage

Storage of performance data in SisMonit is based on event classes. Event classes abstract activities or exceptions of the monitored component, such as alarms and management actions. Each event has a set of attributes that are inherent to the event class and may be configured in according to monitoring needs and intrinsic characteristics of the monitored event. Each distinct set of attributes and their values identify an instance of the class. SisMonit keeps a performance profile for each instance, which stores information such as number of occurrences and cumulative duration of the events that fall in the instance. The instances of each class are organized as a tree, where the each level considers an attribute as branch criterion, and the assignment of attributes to levels is configurable. Thus, the monitoring process produces a forrest of profile trees, where each tree stores the profiles of a class of events.

The profiles are also accumulated hierarchically, i.e., the measures of each instance are also summarized in their tree ancestors, and the root of each sub-tree stores performance data for all nodes in the sub-tree. This cumulative performance data is kept in a second set of node variables, so that both cumulative and non-cumlative data are available for analysis.

3.1.2 Monitoring Level

Measurement data can be divided into three levels of detail, which provide an increasing amount of information about the monitored application: (1) counters; (2) timers; and (3) log events. SisMonit supports these three measurement levels, but it is usually hard to determine, upfront, the level that will be employed during the component's life cycle. Moreover, it is usually necessary to vary the level of detail in order to accomplish activities such as performance tuning. Therefore, we chose to provide a flexible monitoring mechanism that may acquire performance data at various levels of detail from the same instrumented code. Details about the measurement level are hidden inside the monitoring library, and the measurement level is set in the configuration file.

As mentioned before, attributes are module specific and also configurable from a file. Although some management platforms have pre-defined monitoring hierarchies (i.e., MIBs), we believe that it is hard to define these hierarchies a priori, as a consequence of the system dynamic behavior – frequent insertions and deletions. Besides, code instrumentation is flexible by definition, since it is performed by the module programmer himself.

3.1.3 Acquisition

As mentioned, SisMonit supports the acquisition of three types of performance data: (1) counters, (2) timers, and (3) log records. Counters are accumulators that are incremented whenever the associated event occurs. Counters are a low-cost strategy, since it is not necessary to perform any system call to register an event. Timers are similar to counters, but store the cumulative duration of a class of events and not only the number of occurrences. They are more intrusive than counters, since they require system calls in order to read the system clock. Log record writes the events and related information (e.g., attributes, starting and ending time) to the monitoring output file. It is the most intrusive data acquisition strategy, as a consequence of system calls for not only reading the system clock, but also writing the records.

Given the task module variety, it is hard to define a single measurement granularity for all applications and classes of events that should be monitored. Notice that measurements can not be the same for all modules, because of their differences and monitoring needs for each application.

The acquisition of event performance data is comprehensive regarding strategy, i.e., whenever we employ timers, counters are also kept, and whenever we employ log records, both timers and counters are kept.

3.1.4 Notification

Monitoring information notification may happen on demand (i.e., the monitored process answers a message that requests for information) or periodic (i.e., interval, number of events, filled buffer). The notification strategy, as other parameters, is also defined in the configuration file. An example of monitoring output file can be seen in Figure 3.

```
Mar 31 10:24:18 picard 61704.monit: 0 8267 P Alarm_MFDC  1539 0 0.000000 2102.3
01063
Mar 31 10:24:18 picard 61704.monit: 0 8280 P Alarm_MFDC  Entity="picard"  342
 0 0.000000 1855.973507
Mar 31 10:24:18 picard 61704.monit: 0 8281 P Alarm_MFDC  Entity="picard"
Protocol_type="PING"  0 171 1771.487818 0.000000
Mar 31 10:24:18 picard 61704.monit: 0 8282 P Alarm_MFDC  Entity="picard"
Protocol_type="RPS"  0 171 84.485689 0.000000
Mar 31 10:24:18 picard 61704.monit: 0 8283 P Protocol_replay  171 0 0.000000
53.997456
Mar 31 10:24:18 picard 61704.monit: 0 8284 P Protocol_replay  Entity="picar
d"  171 0 0.000000 53.997456
Mar 31 10:24:18 picard 61704.monit: 0 8285 P Protocol_replay  Entity="picar
d"  Protocol_type="PING"  0 171 53.997456 0.000000
```

Figure 3: Monitoring Output Sample

3.2 Instrumentation Interface

In this section we present the monitoring interface of SisMonit. This interface is also comprehensive, since it allows monitoring at the various levels. It also aims to minimize the need for frequent code instrumentations or changes in existing instrumentations, and is flexible regarding data granularity and the amount of detail about the events.

Since instrumentation of SIS modules is performed explicitly and based on library functions, the monitoring granularity may be as coarse (or thin) as needed. As discussed, the number and nature of attributes vary among classes of events and are also configurable. This attribute-related flexibility is also supported by the proposed interface through an attribute definition scheme based on comparisons between strings, making it virtually unlimited. Each attribute is identified by a string and may be of the following types:

Integer: 32-bit integer.

Real: 64-bit floating point number.

String: string delimited by the null character.

Date hour: formatted string containing a clock value up to microseconds, e.g., `1999:-03:31:08:04:23:236876`.

Time: time interval in seconds and microseconds.

The monitoring interface is composed by the following primitives:

initmonit(arqmonit): reads the configuration file **arqmonit** and initializes the monitoring data structures.

evid = eventstart(class, parent): sets the beginning of an event from class **class** that should be referenced by **evid**. If **parent**, another event being also monitored, is not null, all of its attributes are inherited.

eventattr(evid, id, tipo, valor): defines an attribute identified by **id** of event **evid**, setting the value **valor** that has type **tipo**.

eventstop(evid): suspends temporarily the monitoring of **evid**, applicable to log records and timers.

eventresume(evid): resumes the monitoring of **evid**, applicable to log records and timers.

eventend(evid): indicates that event **evid** terminated and causes the associated profiles to be updated.

eventreset(evid): initializes the data structures associated with **evid**.

eventget(evid,med): returns the current monitoring measures associated with event **evid** in **med**.

eventflush(evid): prints event **evid** to the monitoring output file.

Finally, notice that, by using this interface, it is possible to monitor events that overlap partially or totally, without any instrumentation changes.

4 Using SisMonit

After instrumenting the module code, the generation of performance profiles is specified through configuration files. This section describes the possible settings and how diverse monitoring demands can be satisfied. The configuration file contains the following information: *i)* measurement strategy; *ii)* notification strategy; *iii)* event classes and attributes that should be monitored; *iv)* supervision predicates; *v)* filters; and *vi)* output files.

All information is stored in a configuration file that is loaded by the monitored module. In the sections that follow we describe the two entities that are specified in the configuration file: event classes and monitoring predicates.

4.1 Event Classes

As defined in Section 3.1.1, event classes abstract modulés activities or exceptions that we want to monitor. Each class attribute is characterized by an identifier, its type, and a name. Event classes may be organized as hierarchies, inheriting attributes of other classes. There are two types of statements for defining classes in configuration files:

Class Definition: defines the class identificator and its ancestral class, if there is any. This statement starts with the string `@C`.

Attribute Definition: defines an attribute, through its type and name. This statement starts with the string `@A`.

```
@C Alarms
@A Alarms Entity string
@C Alarm-MFDC Alarms
@A Alarm-MFDC Protocol string
@C Protocol-replay Alarm-MFDC
@C Trap-received Alarms
@C Date
```

Figure 4: Examples of Classes and Attributes

Figure 4 presents the description of five classes (`Alarms`, `Alarm-MFDC`, `Protocol-replay`, `Trap-received` e `Date`). `Alarms` contains only `Entity` which is a `string`. The classes `Alarm-MFDC` and `Trap-received` are derived from `Alarms`, inheriting `Entity`. `Alarm-MFDC` has another attribute, `Protocol-type`, that is also a string. Finally, `Date` does not have any attribute.

4.2 Monitoring Predicates

Monitoring predicates specify how the monitoring should be performed from six perspectives: *i)* event classes or sub-classes; *ii)* monitoring strategy (counter, timer ou log record); *iii)* notification frequency (e.g., at the end of module's execution, periodically); *iv)* monitoring output file; *v)* monitoring conditions; and *vi)* attributes that are written in the output file.

There is one predicate per line (which starts with the string `@P`) specifying the event class, monitoring strategy, notification frequency and output file. The conditions that have to be satisfied for enabling the monitoring are expressed through a logical expression, which is delimited by parenthesis, being a sequence of term conjunctions (`&&`), disjunctions (`||`) ou negations (`!`). Each term is composed by a relational operator `==`, `!=`, `<`, `>`, `>=` e `<=`) that establishes a relation to be satisfied between literals or variables. Besides class attributes, expressions may contain monitoring internal variables, which are listed in Table 1.

Figure 5 presents three examples of predicates. The first predicate specifies that we will keep counters for all classes; notifications occurr every other first minute and second in the file `/log`. The second predicate specifies that events from the class `Alarms` that

Id	Strategy	Meaning
`@DH`	All	current date and time
`@CT`	All	counter cumulative value
`@TI`	Timer and Log record	timer value
`@TM`	Timer and Log record	timer cumulative value
`@DE`	Registro	event duration

Table 1: Monitoring internal variables

occur between 7:50 and 8:00 AM daily are timed; notification is performed on the file `/log` as soon as the file buffer fills, and the `Entity` is printed. The third predicate generates records immediately for events from the class `Alarm-MFDC` whose `Protocol-type` is equal to `PING` and duration greater than two seconds; the record is generated in the file `/log`.

```
@P * counter ::::00:00 /log ()
@P Alarms timer buf /log ((@DH>:::07:50:00) && (@DH<:::08:00:00)) [Entity
@P Alarm-MFDC trace agora /log ((@DE>2.0)&&(Protocol-type=="PING")) [En
tity Protocol-type]
```

Figure 5: Monitoring Predicates Sample

5 Performance Monitoring of a agent

5.1 The agent/manager model

The generic interface between SIS and network elements follows the manager-agent model [5]. A SIS-agent has three main modules: (1) `MisUser`, (2) `LibAgente` and (3) `agent-specific code`. The MisUser, *Management Information Service User*, provides a communication interface between the agent and the manager. It uses management systems services. The Libagent is a library of functions that get information from a configuration file and provides communication primitives between the agent and the `MisUser`. Agents use some of these functions (e.g., AgentInitialize, ReceiveResponse, GetAlarmsList). The general agent behavior is defined by `Libagent`, which also contains the main function of the agent. This module provides a standard way to build agents, abstracting all communication details. Besides, it also provides several support routines.

The specific agent module implements some routines related to network element, such as parsing, mapping and physical communication with the network element. For each new agent that employs Libagent, a specific module has to be implemented.

5.2 Monitoring the Agent MFDC

In order to demonstrate the use of SisMonit, we chose the MFDC agent, which was implemented using the agent-manager model to supervise workstations and communication equipments that use TCP/IP protocol. This agent employs RPS (*remote ps*), SNMP (*simple network management protocol*) and ICMP (*ICMP-ECHO - ping*) to communicate with the network elements. The communication between the agent and SIS employs the Libagent primitives. Workstation supervision gets information about its availability, memory usage, and per-process memory usage.

The supervision process consists of querying network elements using one of the protocols described above. If the network element is a workstation then we can use the PING and RPS protocols. The agent gives information whether a workstation is communicating with all other hosts in the net by using the PING protocol. Information about memory usage is available when the agent uses the RPS protocol. If the network element is a router, the SNMP protocol is used, allowing the agent to get information about its interface's state. The querying process ends when the agent notifies its manager.

Each agent's configuration is stored in the SIS database, which contains a list of the network elements to be supervisioned and the respective communication protocol to be used. Besides, the agent has information about the query interval and its manager. When an agent process starts, configuration information is recovered by Libagent functions from the Data Base. After determining which protocol it will be used to gather data from the network element, a function from Libagent is called to initialize the physical layer communication. Then, the agent enters in a loop where it blocks until it is time to query the element, when the function GetAlarmsList is called. The agent stays in this loop forever.

The performance of the MFDC agent is monitored by the SisMonit library (described in Section 3). The MFDC agent code was instrumented as well in the Libagent code. At the Libagent, the monitoring goal was to determine the elapsed time between alarm detection by an agent and its registration in SIS. A sample configuration file is presented in Figure 4

5.3 Monitoring

The monitoring functions were inserted in the Libagent code as well the MFDC agent code. We used the following event classes:

Alarms: used to monitor all alarms of SIS agents. This class was inserted at the Libagent code. It has one attribute: Entity.

Alarm-MFDC: used to monitor only MFDC alarms. It's derived from the class Alarms, inheriting the attribute Entity. It has another attribute: Protocol-type.

Protocol-replay: used to monitor the alarms that describe communication failures between the agent and the supervised networks elements. It's derived from the class Alarm-MFDC inheriting the attributes Entity and Protocol-type.

Trap-received: used to monitor the function Trap-received from the agent's code.

Date: used to monitor the function Date from the agent's code.

The two attributes have the following meaning:

Entity: defines which is the supervisioned network element.

Protocol-type: defines the protocol used at the MFDC agent. The possible values of this attribute are PING, RPS, and SNMP.

5.4 Performance Metrics

As mentioned, both Libagent and the agent itself were instrumented for gathering performance information. Libagent monitoring provides the following information: *i)* time to query the network element, detect occasional faults, and insert the proper alarm. *ii)* number of generated alarms.

On the other hand, the agent-MFDC monitoring provides the following information: *i)* When the communication between the agent and the network element has failed, and how much time has passed since its detection and insertion into SIS alarms list. *ii)* How frequent router interfaces had failed and the workstations had a high memory usage, and how much time has passed since its detection and insertion into SIS alarms list. *iii)* What the cost of querying a socket for incoming SNMP packets is. *iv)* What the cost to calculate the alarms date is. *v)* What the number of generated alarms by the agent-MFDC is.

6 Remote Supervision

Our second monitoring example involves the agent MICRODX, which is also based on the agent-manager model. This agent uses a centralized program to perform the communication with the network element (or remote unit) called PGR (Remote Manager Program). PGR provides real time telemetry and telecommand operations from/to the monitored devices. These devices are associated with objects (nodes and variables). Variables are numerical entities and Node is a binary entity. Each agent configuration, which is stored at the SIS database, has a list of the stations (remote which are the monitoring devices) that will be supervised. The alarm acquistion is performed as follows: (1) the agent sends a command to PGR asking for communication establishment between the agent and a station; (2) after the communication is set, it sends the alarms updates to the log file; (3) the agent gets the current alarm list; and (4) the agent sends to SIS all alarms using Libagent. This process is common to all monitored stations.

6.1 Monitoring

Similarly to MFDC agent, both Libagent and Microdx were instrumented. A partial monitoring configuration file is showed in Figure 6. We used the following classes of events:

Alarms: used to monitor all alarms of SIS agents. This class was inserted at the Libagent code. It comprises two attributes: Entity and Description.

Alarm-Microdx: used to monitoring only Microdx alarms. It's derived from the class Alarms, inheriting the attributes Entity and Description. It has another attribute: Object.

The three attributes have the following meaning:

Entity: used to define the supervisioned network element.

Description: contains the description of the alarm.

Object: defines object type, Node or Variable.

The performance metrics are basically the same of the MFDC agent.

```
@C Alarms
@A Alarms Entity string
@A Alarms Description string
@C Alarm_Microdx Alarms
@A Alarm_Microdx Object string
@p aux_tmp timer fim /var/adm ((@ct>55)&&(@ct<44)) [sev]
```

Figure 6: Microdx Configuration File

7 Conclusions and Future Work

In this paper we presented SisMonit, the performance monitoring system of the Integrated Supervision System. This monitoring system is characterized by high flexibility and multiple functionalities, allowing different acquisition granularities and types of performance data that can be collected. We also presented a usage example, where two modules, an agent and a manager, were monitored.

We expect this monitoring system to be employed throughout the whole SIS platform for performance management, allowing operators and system designers to monitor not only each module in isolation, but also their interaction, identifying global bottlenecks.

References

[1] H. Andrade, J. Nogueira, and A. Loureiro. On the experience of using software components in a telecommunications network management project. In *Proc. of International Conference on Telecommunications - ICT98*, 1998.

[2] W. Meira Jr. *Understanding Parallel Program Performance Using Cause-Effect Analysis.* PhD thesis, Dept. of Computer Science – University of Rochester, Rochester, NY, July 1997. Available as TR 663 – DCS – University of Rochester.

[3] W. Meira Jr., T. LeBlanc, and V. Almeida. Using cause-effect analysis to understand the performance of distributed programs. In *Proceedings of SPDT98: SIGMETRICS Symposium on Parallel and Distributed Tools*, Welches, OR, August 1998. ACM.

[4] José Marcos NOGUEIRA and Dilmar Malheiros MEIRA. The SIS Project: A Distributed Platform for the Integration of Telecommunication Management Systems. RT SIS 3107, UFMG-DCC-ICEx, Belo Horizonte-MG, september 1995.

[5] José Marcos S. NOGUEIRA, Patrícia V.C. BICALHO, Murilo S. MONTEIRO, and João E.R. DANTAS. Interfaceamento com Elementos de Rede: Especificação da Interface Genérica com o SIS. RT SIS 3104, UFMG-DCC-ICEx, Belo Horizonte-MG, agosto 1994. 1^a edição.

[6] R. Silva and H. Andrade J. Nogueira. Methodology and experience in bulding powerful telecommunication network management agents. In *Proc. of IFIP/IEEE Distributed System Operation and Management - DSOM 96*, 1996.