Objective-Driven Monitoring for Broadband Networks

Subrata Mazumdar and Aurel A. Lazar, *Fellow, IEEE*

Abstract—An approach to sensor configuration, installation, and activation for real-time monitoring of broadband networks for managing its performance is presented. An objective-driven measurement strategy for establishing the dynamic and statistical databases of the network is described. Objective driven monitoring allows the activation of sensors for data collection and abstraction based on a set of objectives. The objectives are derived from the quality of service requirements for real-time traffic control and operator submitted queries. The methodology of objective-driven monitoring for selective activation of sensors is implemented as a set of rules in the knowledge base of the monitor.

Index Terms—Network, quality of service, performance management, sensor, monitoring, knowledge-based systems.

1 Introduction

BROADBAND networks consist of many subsystems (switching nodes, multiplexers, links, etc.) that are geographically distributed, carry multiple classes of traffic and have access to different information patterns. Although these subsystems make their own local decisions, they work together for the achievement of the common system wide goal of information transport. The common goal is to guarantee the Quality of Service (QOS) negotiated during the call setup for each of the traffic classes [1]. The QOS is specified through a set of performance parameters.

Monitoring of these parameters and of all network resources, such as buffer space, switching and communication bandwidth, and call processing, is required in order to guarantee the QOS [1]. A network monitoring system should also be applicable to several representative networks. Therefore, a proposed set of measurement parameters must be network independent [2]. They must be declared in generic terms, such as throughput, time-delay, arrival rate, inter-arrival time, etc. Sensors (measurement points) for these parameters must be made available in all the networks to be monitored. A set of objective criteria or strategies are needed by which sensors can be selectively activated and deactivated among a large number of sensors in a distributed environment. One of the main objectives of the monitoring task is the real-time support of the network control and management system during the decision making process. A consistent view of the network is assumed to be available for monitoring [3].

The monitoring of networks can be viewed at different levels of abstraction. Monitoring takes place both at hardware and software level depending upon the hardware and software components that support the information transport. In [4], a network operation center to monitor, control, and manage ARPAnet-like packet-switching networks is presented. In [5], [6], [7], and [8], network monitoring is done for LANs or interconnected LANs carrying only single class (data) traffic. In the latter work, major emphasis was on the evaluation of usage of *communication resources*. In [9], monitoring of a metropolitan area network, called MAGNET II, is carried out by hardware observation units (HOU) connected to network access points. Real-time traffic measurements are reported. The quality of service of traffic classes in the network is evaluated by monitoring the buffer occupancy distribution, the packet time delay distribution, the packet loss, and the gap distribution of consecutively lost packets. In [10], the monitoring of switching resources was considered for managing AT&T's dynamic nonhierarchical routing algorithm for automatic as well as operator oriented control of the network.

Since a network can be considered to be a distributed system, the approaches to monitoring of distributed systems can also be applied to monitoring broadband networks. The monitoring of distributed systems can be classified as event-driven monitoring and as a database approach to monitoring. Most of the work in event-driven monitoring of distributed systems was done on the application level. Debugging of distributed systems [11], [12], [13] and parallel programming environments [14] are typical examples. Here, major emphasis was given to the performance evaluation of *processing resources*. In [15], a relational approach to monitoring was presented. In the relational approach, monitoring is viewed as an information processing activity and the *historical database*, a class of relational databases that encode time, is considered an appropriate formalization of the information processed by the monitor.

In this paper, the steps required to configure, install and activate sensors for monitoring broadband networks are discussed and a knowledge-based approach is presented as a solution to the problem. In order to monitor object behavior, sensors need to be configured and installed in the

- *S. Mazumdar is with Bell Laboratories, 101 Crawfords Corner Road, Holmdel, NJ 07733-3030. E-mail: mazum@bell-labs.com.*
- *A.A. Lazar is with the Department of Electrical Engineering and the Center for Telecommunications Research, Columbia University, New York, NY 10027. E-mail: aurel@ctr.columbia.edu.*

Manuscript received Sept. 16, 1991.
For information on obtaining reprints of this article, please send e-mail to: transkde@computer.org, and reference IEEECS Log Number K96031.

network. Sensor configuration specifies the characteristics of sensors declared in the knowledge database of the monitor. These characteristics are specified by a set of attributes and a set of procedures for operations. Sensor installation involves identification of the measurement points in the network.

The architecture of our objective-driven monitoring system is knowledge-based. It consists of a knowledge database and an inference engine for reasoning on the database [16]. The inference engine consists, in turn, of two parts: a deductive inference processor and a statistical inference processor. The role of the deductive inference processor is to process the queries about the network behavior and activate sensors in the network. The role of the statistical inference processor is to abstract the information obtained by the sensors.

The monitoring system processes queries on the system as well as on the conceptual level, and sets up sensors to collect information. The system level monitor supports queries only if precise knowledge about the system is available. On the conceptual level, the monitor allows general queries without the precise knowledge of the system architecture of the network.

An objective-driven monitoring scheme is presented that selectively activates and deactivates a subset from among a large number of sensors already installed in the network. The objective-driven monitoring scheme is closely related to the concept of *experimental frame* of [17] that characterizes modeling objectives by specifying the form of experimentation required for obtaining answers to the questions of interest. For the class of objective-driven monitoring tasks considered in this paper, the fundamental concepts are derived from the requirements of supporting Quality of Service and of operator submitted queries. The objective-driven monitoring scheme deals with the problem of complexity in monitoring broadband networks through the concept of observation frame that we have earlier proposed [18].

An object-oriented definition of sensors is introduced and a method for specifying the configuration of the sensors in the network is given. This definition represents an alternative to "a collection of code" given in [15]. Through the specification of object-specific and variable-specific generic sensors, we can define the starting and stopping time for monitoring and also how frequently the observation samples are to be collected and recorded. Since the various measures for performance analysis are specified through a set of operators, we can easily add a new set of performance measures or selectively activate a subset of measures. Based on our approach, we can select any object, state variable, event or their performance parameter for monitoring.

This paper is organized as follows. Section 2 outlines the architecture of the experimental environment that represents a platform for knowledge-based monitoring of broadband networks. Section 3 describes the key ideas about sensor configuration, installation and query analysis for monitoring. Finally, in Section 4, the objective-driven measurement strategy and a query based activation of sensors for broadband networks are discussed.

2 The System Architecture of the Monitor

The architecture of the knowledge-based monitoring system was modeled as a real-time system (as shown in Fig. 1) where the monitor asynchronously interacts with the network through an interface [19]. Thus, the network can be viewed as the environment for the monitor. The network behavior is monitored through the interface and the collected information is sent to the monitoring system that maintains an image of the network. The interface is all the monitor sees of the network. The characteristics of the interface depend to large extent on the environment. What is and what is not part of the interface depends on the specific requirements of the management and control tasks.

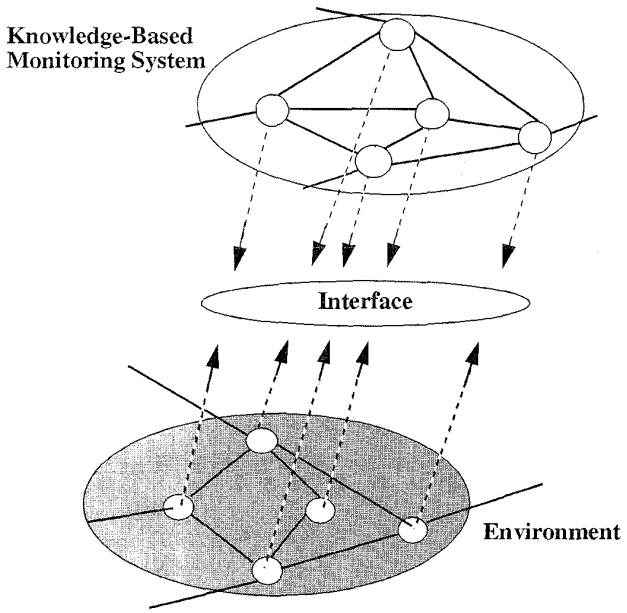

Fig. 1. Real-time system model for monitoring of integrated networks.

The interface between the network and the monitor consists of a set of state variables. A state variable is persistently present and throughout its existence it has a value that changes with progression of time. For the task of monitoring the network, state variables get their values from the processes operating in the environment. The semantic information about network objects and the interface are represented by the Entity-Relationship model [20]. The computational model consisting of a set of sample path and performance evaluation operators defined in [18], [19] is used to describe various processes that are associated with state variables.

Thus, in representing the environment and the interface, the concept of modularity was achieved through the *object* representation. The location and ownership of a state variable was declared through these objects. These objects are responsible for acquisition, manipulation, and dissemination of the information of their state variables. Note that, while implementing the network architecture, one has to explicitly declare a set of state variables that form the interface between the network and the monitor. These variables

characterize the observable behavior of the network. The exact specification of the interface depends on the monitor and the specific management tasks, such as performance, fault and configuration management, that the monitor is going to support.

The architecture of the knowledge-based monitoring system (as shown in Fig. 2) consists of the knowledge database and an inference engine for reasoning on the database for query processing, sensor activation, and interpretation of data collected by the sensors. The inference engine consists of two blocks:

1) the *deductive inference processor* and
2) the *statistical inference processor.*

The role of the deductive inference task is to set up a distributed *observation frame*, i.e., a data space in which a query may be answered.

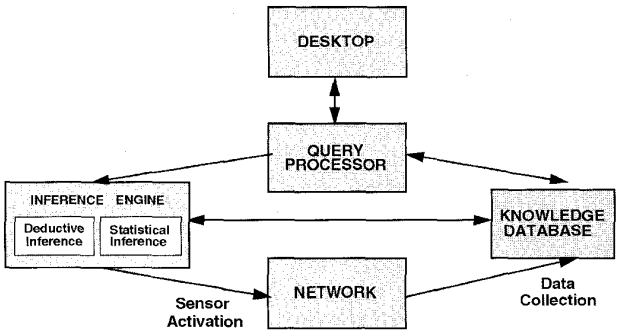

Fig. 2. The system architecture of knowledge-based monitoring.

Fig. 3 shows the organization of the knowledge database of the network. The knowledge database is organized as follows. The system level knowledge about the network is represented in the configuration database. The configuration database contains the knowledge about the network entities, such as buffers, sources, and servers and their specific instances. Fig. 4a describes the attributes of those network entities. The dynamic database contains the information about the state and event variables of the objects in the configuration database. Fig. 4b describes the attributes of the state and event variables. The sensor database contains the generic description of sensor objects and also any specific class of state and event variables and all of the instances of the sensor object class. The objects in the sensor database indicate the specific sampling pattern for data collection and specific sensor instances indicate the activation of the sensors. The sensor database together with the configuration database forms the static database. These two databases change much less often than the dynamic database. The dynamic database only exists for those state and event variables that are measured by activating the sensors in the network. The statistical database is obtained by applying abstraction operators on both state and event variables and provides various performance measures for each state and event variable. Fig. 4c describes the attributes of the performance parameters.

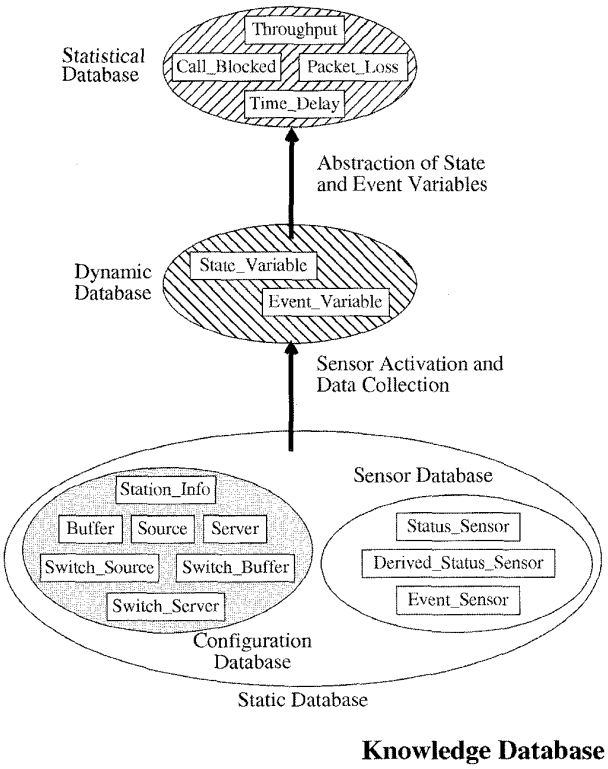

Fig. 3. The organization of the knowledge database.

```
(defschema    NETWORK-OBJECT
    (generic_name)          ;;;; Class name of the Object
    (key)                   ;;;; Composite attribute defining the key attribute
    (status)                ;;;; Indicates the status of object, active or inactive.
)

(defschema    BUFFER
    (IS-A                   NETWORK-OBJECT)     ;;;; Inherits the attribute.
    (generic_name           BUFFER)
    (buffer_id)                                 ;;;; Key attribute of Buffer.
    (buffer_size)                               ;;;; Size of buffer
)
(defschema    SERVER
    (IS-A                   NETWORK-OBJECT)
    (generic_name           SERVER)
    (server_id)
    (server_status)
)
(defschema    SOURCE
    (IS-A                   NETWORK-OBJECT)
    (generic_name           SOURCE)
    (source_id)
    (source_status)
)
```

Fig. 4a. The attributes of basic network objects.

Typically, a query submitted by the query generator, i.e., a control task or a human operator, requires information about the performance of certain objects in the network. The query is then processed to find out the specific instances of the performance parameters of interest. Based upon these parameters, the deductive inference processor creates a derived object containing the identified performance parameters and their corresponding state and event variables. An associated derived sensor capable of monitoring the derived object is also created, which in turn cre-

ates the appropriate sensors in the sensor database for the selected state and event variables. Creation of the sensors for state variables in the knowledge database activates the sensors in the network and data is collected. The statistical inference processor then applies the statistical operators, passed during the query submission or implicit in the performance parameter specification, on the collected information and transmits the processed information to the query processor.

```
(defschema      STATE_VARIABLE
      (IS-A                          NETWORK-OBJECT)
      (generic_name                  STATE_VARIABLE)
      (VAR-OF-GENERIC-OBJECT)        ;;; the class name of object to which it belongs
      (range_of_var)                 ;;; list indicating the upper and lower bounds
      (state_threshold)              ;;; threshold value of state variable

      (value)                        ;;; the value at a sample instant
      (obs_time)                     ;;; the time instant of the sample

      ;;; the following attributes represent operators on state variable

      (min_value)                    ;;; recorded minimum over a period
      (max_value)                    ;;; recorded maximum over a period
      (up_count)                     ;;; number of times threshold is crossed from below
      (down_count)                   ;;; number of times threshold is crossed from above
      (first_obs_time)               ;;; the starting time sampling window
      (last_obs_time)                ;;; the time of observation of last sample
      (top_hit)                      ;;; number of times upper bound is reached
      (succ_top_hit)                 ;;; number of times upper bound is reached successively
      (bottom_hit)                   ;;; number of times lower bound is reached
      (succ_bottom_hit)              ;;; number of times lower bound is reached successively
)

(defschema      EVENT_VARIABLE
      (IS-A                          STATE_VARIABLE)
      (generic_name                  EVENT_VARIABLE)
      (event_operator_id)                ;;; the id of the operator the extracts the event
)
```

Fig. 4b. The attributes of state and event variables.

```
(defschema      PERF-PARAMETER
      (IS-A                          NETWORK-OBJECT)
      (PERF-OF-GENERIC-VARIABLE)
      (PERF-OF-GENERIC-OBJECT)
      (generic_name                  PERF-PARAMETER)

      (count)                        ;;; Total number of samples collected for abstraction
      (first_obs_time)               ;;; start time of the measurement interval
      (last_obs_time)                ;;; end time of the measurement interval

      (min_value)                    ;;; minimum value of collected samples
      (max_value)                    ;;; maximum value of collected samples
      (av_value)                     ;;; average of the collected samples
      (var_value)                    ;;; variance of the collected samples
      (time_wtd_av_value)            ;;; time weighted average of samples
      (time_wtd_var_value)           ;;; time weighted variance of samples
      (min_value)                    ;;; minimum value over measurement interval
      (max_value)                    ;;; maximum value over measurement interval
      (av_top_hit)                   ;;; average of number of times upper bound is hit
      (av_succ_top_hit)              ;;; average of number of times upper bound is hit successively
      (av_bottom_hit)                ;;; average of number of times lower bound is hit
      (av_succ_bottom_hit)           ;;; average of number of times lower bound is hit successively
)
```

Fig. 4c. The attributes of performance parameters.

3 Monitoring

As mentioned in the previous section, the monitor is defined to be a real-time system that maintains an ongoing relationship with its environment, i.e., the network. The interface between the monitor and the network is defined by a set of state variables. The *interface* is all the monitor sees of the network. Thus, the characteristics of the interface depend to a large extent on the network.

Depending upon the information requirement, a network can be monitored in two ways: *monitoring the states* (status monitoring) and *monitoring the events*. During status monitoring the collection of values of any of the state variables obtained by sensor activation is recorded. The rate at which the information is generated by sensors depends on the speed of operation of the corresponding objects, such as buffers, servers, and sources. For example, on the network access level, the rate of generation of state information may be equal to the packet arrival and departure rate; and at the session layer, it may be equal to the rate of arrival of new calls. Events are abstractions of state variables obtained by applying the sample path operators of the computation model over time. An event derived from a state variable is recorded by a sensor as an event variable.

The design of the network monitor can be characterized by the following steps that are in part, adapted from [15]:

- Step 1: Sensor Configuration.
  This step involves the design of the sensor, i.e., specification of its attributes and the procedures of operation that handle the necessary interaction with the monitor, enabling and disabling the sensors and buffering of monitored data and requested tasks. The sensor attributes specify the starting and stopping times for monitoring, how frequently the monitored events, measures, or resources are to be recorded, and other related performance information to be collected simultaneously;
- Step 2: Sensor Installation.
  This step involves identifying the state and event variables that are to be monitored by sensors.
- Step 3: Query Analysis Specification.
  This step specifies how to decompose a query, activate sensors and create various dataspaces for information abstraction.
- Step 4: Execution.
  This step is comprised of activating the sensors, generating and abstracting the data collected from the network, transmitting the data from the network to the monitor, and finally presenting the data on a graphics terminal. (This step is discussed in Section 4.)

Even though we have adopted the steps in [15], there are differences between [15] and our approach. In [15], the approach was relational whereas our approach is object-oriented. Since a sensor monitors state variables, there exists only one type of sensors that need to be configured for installation in the network. Thus, we need to instrument an object only once in order to obtain various measures, such as average and variance of buffer occupancy from the buffer state variable associated with a buffer. In our approach the various performance measures are defined as operators for the sensor.

In [15], there is no concept of object-specific or variable-specific generic sensors. Through the inheritance mechanism of the object-oriented approach we can specify object or variable specific generic sensors. In the case of sensor installation, we show how to select the measurement points based on performance management objectives. These measurement points are based on the actual variables that are responsible for generating information and on which the performance measures are applied. In the query analy-

sis specification step we show how the data transformation takes place during data collection and how the interpretation is done based on this data. We have a clear separation between the raw data that is collected and its abstractions. In this step, we show how the information provided by a simple query is used to identify sensors, collect data, detect events and how abstraction and interpretation is done on the collected data.

3.1 Sensor Configuration

A *sensor* is defined to be an object with a set of attributes and a set operators (algorithm or code), implemented either in hardware or software. The sensors installed in the network collect information about the state or event variables of an object and transfer it to the monitor. From an implementation point of view, every state and event variable includes its sensor as a component object. Sensor operations are executed by the set of sample path and statistical operators described in [18], [21].

Sensors installed in the network to monitor the state variables of an object are termed *primitive sensors*. The primitive sensor corresponds to the object class SENSOR in the sensor database; its attributes are the same as that of SENSOR (as shown in Fig. 5). The attribute **sensor_code_id** specifies the operator to be applied to abstract information from the history of a state variable. The attribute **initiated_by** indicates the initiator of the query based on which the sensor is activated. Primitive sensors contain the code for sample path and performance evaluation operators.

The attributes of a sensor are defined based on the requirements for both status and event monitoring and they are shown in Fig. 5. The abstraction operators of a primitive sensor operate on two time scales. The sample path operators abstract events on the time scale of the state variables. The performance evaluation operators operate on both state and event variables on a time scale based on the interval for statistics collection. The parameters for performance evaluation operators are provided by a set of sensor attributes. These attributes are **sample_count**, **sample_on**, **sample_off**, **duration_of_activation**, and **sampling_interval**.

The sensor attribute **sample_count** specifies the total number of state and event variables samples that are to be collected for statistical inference. The average on a fixed number of samples is computed based on the value of the **sample_count** and it is the default procedure for evaluating the average over a period. If the value of **sample_count** is not specified and the attribute **sample_on** is specified, then the later is used with **sampling_interval** to compute the total number of samples to be collected for statistical inference. The specific values of **sample_count** or **sample_on** for monitoring a state variable are determined by the rate at which the variables are changing their values. Their values are also determined by the control algorithm that is managing the object. In order to repeat the statistical inference process, the sensor attribute **duration_of_activation** specifies the duration of the monitoring process or the duration of time the sensor remains active. The attribute **sample_off** of sensor specifies the duration between two consecutive measurement intervals, i.e., **sample_on** period.

```
(defschema      SENSOR
    (IS-A                       NETWORK-OBJECT)
    (generic_name               SENSOR)
    (sensor_code_id)                        ;;; code for abstraction operator
    (initiated_by)                          ;;; user query or program query
    (duration_of_activation)                ;;; how long the sensor remains active
    (sample_on_period)                      ;;; length of window open for monitoring
    (sample_off_period)                     ;;; time between two succesive windows
    (sampling_interval)                     ;;; sampling rate
    (sample_count)                          ;;; number of samples to be collected
)

(defschema      STATUS_SENSOR
    (IS-A                       SENSOR)     ;;; inherits sensor's attributes
    (generic_name               STATUS_SENSOR)  ;;; class name
    (MONITORING-GENERIC-STATE-VAR)          ;;; class name of state-variable
    (MONITORING-STATE-VAR)                  ;;; name of specific state-variable
)

(defschema      EVENT_SENSOR
    (IS-A                       STATUS_SENSOR)  ;;; inherits sensor's attributes
    (generic_name               EVENT_SENSOR)   ;;; class name
    (event_operator_id                      ;;; the operator id for event extraction
    (MONITORING-EVENT-VAR)                  ;;; name of the specific event
)

(defschema      DERIVED_SENSOR
    (IS-A                       SENSOR)     ;;; inherits sensor's attributes
    (generic_name               DERIVED_SENSOR)  ;;; class name
    (DERIVED_FROM)                          ;;; the names of component sensors
    (MONITORING-GENERIC-OBJECT)             ;;; class name of object being monitored
    (MONITORING-OBJECT)                     ;;; name of monitored object
)
(defschema      SENSOR_NETWORK_STATION
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_NETWORK_STATION)
    (key                        (node_no st_no))
    (node_no)
    (st_no)
    (MONITORING-GENERIC-OBJECT  NETWORK_STATION)  ;;; class name of object being monitored
)


(defschema      SENSOR_SWITCH_BUFFER_INFO
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_SWITCH_BUFFER_INFO)
    (key                        (node_no st_no))
    (node_no)
    (st_no)
    (MONITORING-GENERIC-OBJECT  SWITCH_BUFFER_INFO)
)
(defschema      SENSOR_LINK_BUFFER_INFO
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_LINK_BUFFER_INFO) ;
    (key                        (node_no st_no))
    (node_no)
    (st_no)
    (MONITORING-GENERIC-OBJECT  LINK_BUFFER_INFO)
)
(defschema      SENSOR_BUS_SWITCH_FABRIC
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_BUS_SWITCH_FABRIC)
    (key                        (node_no st_no))
    (node_no)
    (st_no)
    (MONITORING-GENERIC-OBJECT  BUS_SWITCH_FABRIC)
)

(defschema      SENSOR_SWITCH_BUFFER
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_SWITCH_BUFFER)
    (key                        (node_no st_no buffer_id))
    (node_no)
    (st_no)
    (buffer_id)
    (MONITORING-GENERIC-OBJECT  SWITCH_BUFFER)
)

(defschema      SENSOR_LINK_BUFFER
    (IS-A                       DERIVED_SENSOR)
    (generic_name               SENSOR_LINK_BUFFER)
    (key                        (node_no st_no buffer_id))
    (node_no)
    (st_no)
    (buffer_id)
    (MONITORING-GENERIC-OBJECT  LINK_BUFFER)
)
```

Fig. 5. The schema description of sensor and its subclasses.

Primitive sensors are activated by sending them a message. Conversely, a primitive sensor transmits information to the monitor by sending a message. Each message is time stamped with the time of creation of the information sent. If the transmitted message contains the value of a state variable then the time of creation indicates the last sampling time. If the message contains an event indication, then the time of creation indicates the event occurrence time. If the message contains the information about a performance parameter of a state or event variable, then the time of creation indicates when the value of the performance parameter was computed.

Primitive sensors are provided with the capability to queue up multiple requests for monitoring. The messages sent by the monitor to the sensor contain information about the specific sample path and performance evaluation operators to be applied to the collected values. In order to allow multiple users or control algorithms to query the state and event variables, the primitive sensors are provided with the ability of both one-to-one and one-to-many communications.

Two subclasses of primitive sensors, STATUS_SENSOR and EVENT_SENSOR, are defined to monitor the state and the event variables, respectively. The STATUS_SENSOR inherits all the attributes of SENSOR. The relationship type MONITORING-GENERIC-STATE-VAR and MONITORING-STATE-VAR establishes the relationships between the sensor and the subclass of a state variable and the specific instance of the state variable being monitored, respectively. The EVENT_SENSOR is declared as a subclass of the STATUS_SENSOR and thus inherits all of its attributes. The EVENT_SENSOR has an additional attribute, **event_operator_id**, that defines the operator for extracting the event. Since the behavior of all objects is represented by their state variables, only one type of primitive sensors needed to be configured. Thus, no matter how complex, an object can be monitored as long as its state variables are declared. Therefore, no object specific sensor needs to be configured.

In order to monitor an object whose behavior is defined by a set of state or event variables, *derived sensors* are defined. The *derived sensors* are an aggregation of a set of primitive and derived sensors. Derived sensors belong to the object class DERIVED_SENSOR, which is a subclass of SENSOR. The DERIVED_SENSOR is obtained based on the primitive sensors associated with the state and event variables of an object to be monitored. An instance of DERIVED_SENSOR created for monitoring an object will contain the corresponding instance of STATUS_SENSOR and/or EVENT_SENSOR. The DERIVED_SENSOR maintains a list of primitive or derived sensors by the relationship attribute DERIVED_FROM. Since the behavior of an object is always expressed by its state and event variables, the sensor that monitors an object is always a member of the subclass DERIVED_SENSOR and is an aggregate object containing the status and event sensors. Thus, in order to monitor the state of a BUFFER, a DERIVED_SENSOR will be created for the BUFFER and it is composed of a STATUS_SENSOR that monitors the state variable representing the BUFFER's state. If an object is an aggregation of a set of objects, the DERIVED_SENSOR for the aggregate object will consist of the DERIVED_SENSOR of the component object.

The DERIVED_SENSOR may be specialized to represent object specific monitoring information such as the sampling pattern. Thus, in order to obtain the behavior of an object NETWORK_STATION, SENSOR_NETWORK_STATION, a subclass of DERIVED_SENSOR, is defined. The relationship between SENSOR_NETWORK_STATION and the corresponding object class, is established by MONITORING-GENERIC-OBJECT and the relationship between the specific instances of the SENSOR_NETWORK_STATION and the specific instance of NETWORK_STATION, which is being monitored, is established by MONITORING-OBJECT-INSTANCE.

Whenever a specific object in the network is to be monitored, an instance of the DERIVED_SENSOR is created in the sensor database. The value of the MONITORING-GENERIC-OBJECT of DERIVED_SENSOR specifies the class name of the object that the derived sensor is monitoring. The instance of the DERIVED_SENSOR and its association with the object in the configuration database is deleted when the sensor is deactivated at the end of the monitoring period. The derived sensors only exist in the sensor database. *Unlike primitive sensors, no counterpart of derived sensor exists in the network.* Both primitive and derived sensor instances are stored in a database called *sensor database*, as shown in Fig. 3.

3.2 Sensor Installation

Sensor installation allows the selection of the measurement points in the network, i.e., the state variables of the network objects that define the interface between the monitor and the network. Since network object and state variables can be uniquely identified in the system, the events associated with state variables and their performance parameters can be selected. In the modeling process of the monitor in [21], state and event variables were identified based on the performance management requirements. The sensor location was determined by the location of the identified state variables in the network.

Along with sensor configuration, the installation of primitive sensors is the only manual process associated with our monitoring scheme. Since the specification of state variables is to be decided based on the performance objectives of the control tasks, these two steps will now be required to be carried out during the specification and design step of the network. This is by itself a manual process. Thus, the design of the network monitor has been shifted to the network design phase. This design process results in a robust network design that requires very little tuning during operations. The identification of the sensor location during the specification phase helps to alleviate the reliability and the correctness problem of sensor operations.

3.3 Query Analysis Specification

Query analysis specification derives levels of abstraction of the collected information for performance analysis. It also selects the performance analysis criteria and algorithms for various performance measures. As shown in Fig. 6a, a transaction for a query has three parts: an identification function (I) that selects the state or event variables of a specific object to be monitored, a data transformation function

(F) for performance evaluation through statistical inference, and an inference rule (R) to be applied to the abstracted data. The dataspace generated by monitoring a state or an event variable is denoted by the circle F_t in Fig. 6a. It is created after activating the sensor associated with a state or an event variable. The data transformation function (F) (derived from the set of statistical operators) is applied on the dataspace F_t to abstract information from the history of a state or an event variable. Application of F generates a dataspace G_t that consists of only statistical information. If the statistical abstraction is not needed then F is reduced to the identity operator. The inference rule R operates on the dataspace of G_t to further evaluate the statistical information, e.g., for event detection through threshold crossing. If no such operation is needed, then the rule R is reduced to the identity operator.

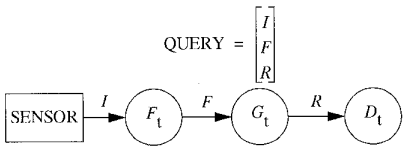

Fig. 6a. Decomposition of a simple query about a state variable.

The deductive inference rule is used to decompose a query into a set of simple queries and then aggregate the information received by servicing the simple queries. For example, in order to find the total average time delay of a call, a set of sensors at the nodes along the route of the call is to be activated to measure the time delay at each node. Once the average time-delay from every node is available, they are aggregated to compute the total average time delay. Therefore, a query for the average time delay of a call will be divided into multiple simple queries and appropriate sensors will be activated to measure the average time delay at every node along the route of the call. Fig. 6b describes such a scheme, where SENSOR_1 through SENSOR_N measure the time delay at each node along the route of the call. The function $f(D_t^1, \ldots, D_t^N)$ in Fig. 6b represents the deductive part of the query for data aggregation and it is applied after the data is collected from the appropriate sensors.

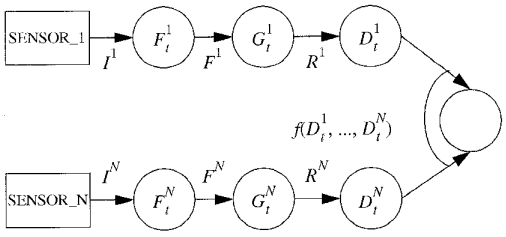

Fig. 6b. Decomposition of a compound query about state variables of multiple object.

We may also want to find out whether a certain average throughput-time delay condition, at a buffer of a node, is satisfied. In this case, we need to activate sensors for both the throughput and the average time-delay and send an event indication if the average throughput-time delay condition is not met. In order to do that, the original query will be divided into two simple queries based on the throughput and the time-delay to be computed. Fig. 6b represents such cases where the original query is divided into multiple simple queries identifying each of the state variables to be measured. $f(D_t^1, \ldots, D_t^N)$ indicates the function that generates an event if the throughput-time-delay condition is not met.

Fig. 6c represents the case when a state variable is monitored for status monitoring and event reporting or multiple event reporting. $f(D_t^1, \ldots, D_t^N)$ represents any deduction to be done after the data is collected. One such deduction scheme is the correlation between two events generated from the same variable. This scheme can also be used to define higher level events based on the history of event variables.

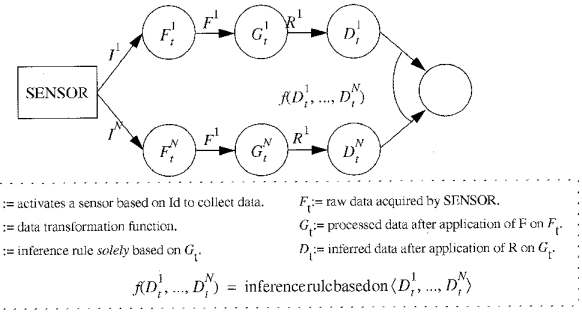

Fig. 6c. Decomposition of a compound query about state variables of same object.

4 The Objective-Driven Measurement Strategy

In the process of performance management, a set of objectives often lead to asking a specific set of questions about the network. The questions of interest could be: Does the network support a specified performance or can the network provide enhanced performance? They can be answered after appropriate monitoring functions are incorporated into the system and the observed data is processed.

Monitoring as a process determined by objectives is called *objective-driven monitoring*. Objective-driven monitoring is closely related to the concept of *experimental frame* of [17] that characterizes modeling objectives by specifying the form of experimentation that is required to obtain answers to the questions of interest.

A query to collect information from the network can be submitted to the monitor either by an user from a terminal or by the various knowledge specialists responsible for network control and management. Such agents are called Query Generators (as shown in Fig. 2). The submitted query can be of two types: real-time data query and non-real-time data query. The real-time data query represents the query on those objects whose attributes are updated using the sensors located in the various subsystems of the network. The nonreal time data query represents the query on those objects whose attributes' value do not depend

upon the sensory information. The nonreal-time queries are handled based on the information available in the knowledge base of the monitor.

The real-time data queries are handled by using the deductive query processing technique, where the inference and retrieval phases of the query have been separated [22], [15]. The schematic of such query processing is shown in Fig. 2. Based on the relationships established in the knowledge database between the various objects, the monitor decomposes the requested query into a set of simple queries and analyzes them to determine specific state and event variables that need to be monitored. Once the state and event variable is identified, corresponding sensors in the network are activated. Activated sensors then collect information about the state and event variables and thus update the dynamic database shown in Fig. 3. If the query requires statistical abstraction of the collected information, then the statistical inference processor applies the corresponding operators on the state and event variables and updates the statistical database.

As described before, one way to find the total average time delay of a call is to activate sensors to measure the time delay at the nodes along the route of the call and then aggregate the average time-delays. A more elegant solution, however, is to first obtain a *derived object* that contains all the state variables that exhibit the average time delay of the intermediate nodes along the route of the call. A derived sensor can be attached to this object and finally adding up the time delays leads to the required result. Thus, to answer a query only a restricted data space, called an *observation frame*, is needed. The observation frame contains only the state and event variables, the performance parameters, and the derived sensor and its components. In the next section a general methodology for an objective-driven measurement strategy is described.

4.1 Deductive Inference

Real-time control algorithms for resource allocation operate based on a set of cost functions and a set of constraints on the behavior of the variables of a system. These system variables could be either describing the state of the system or a statistical abstraction of its state. Thus, a control task first leads to monitoring the system variables. In the case of integrated networks, *QOS parameters* define the target operating points and maintaining these QOS parameters near the operating point becomes the control objective of network operations.

Based on the specified QOS parameters, a set of *performance parameters* are identified. The difference between the QOS parameters and the performance parameters is that the latter depend on the systems architecture of the network. From the specification of the QOS parameters, the corresponding performance parameters are derived by the deductive processor based on the knowledge about the system architecture of the network. The latter resides in the configuration database. As an example, the maximum average end-to-end time delay might be a QOS parameter. The average time delay experienced by a call in a given network is the performance parameter associated with it. It is the aggregation of all the average time delays at nodes and links along the route of the call.

Thus, the request for monitoring a QOS parameter is a query consisting of the class name of an object and the corresponding performance parameter. This general query can be made specific by providing values for one or more key attributes of the object class. Based on the submitted query, the deductive inference processor identifies specific objects and the performance parameters that need to be monitored. First, all the objects in the knowledge base that contain the appropriate performance parameter are identified. Second, the instances of the performance parameters associated with the selected objects are identified. Third, the instances of state and event variables associated with the selected instances of the performance parameters are identified.

For each of the selected objects a component object is created and relationships are established between the new object and the selected performance parameters and the corresponding state and event variables. In order to monitor the selected objects, a derived sensor is associated with the each of the component objects. The creation of the derived sensor generates the instances `STATUS_SENSOR` and/or `EVENT_SENSOR` for each of the state and event variables associated with the component object. Creation of the `STATUS_SENSOR` or `EVENT_SENSOR` activates the corresponding primitive sensors installed in the network. The component object, the associated derived sensor, and the collected information represent a data space called *observation frame*. Thus, the observation frame forms a restricted data space. The answer to queries is obtained by examining, processing, and aggregating monitored information in this space. The statistical inference process takes place only after data has been collected by the sensors.

4.2 An Algorithm for Objective-Driven Monitoring

The operation of the deductive inference processor described above can be formalized into an algorithm consisting of the following steps:

1) identify the instances of the object class specified in the query;
2) identify the instances of the performance parameter specified in the query associated with the selected objects;
3) for each of the selected objects, identify the instances of state and event variables associated with the selected performance parameters;
4) create a new object and associate with it the selected performance parameters and the state and event variables of all the selected objects;
5) associate a derived sensor with the new object and create sensors that monitor the state and/or event variables;
6) activate the sensors in the network;
7) apply statistical inference procedures to evaluate the performance parameter.

The above steps are in part adopted from the "objectives-driven" methodology for modeling of systems [17].

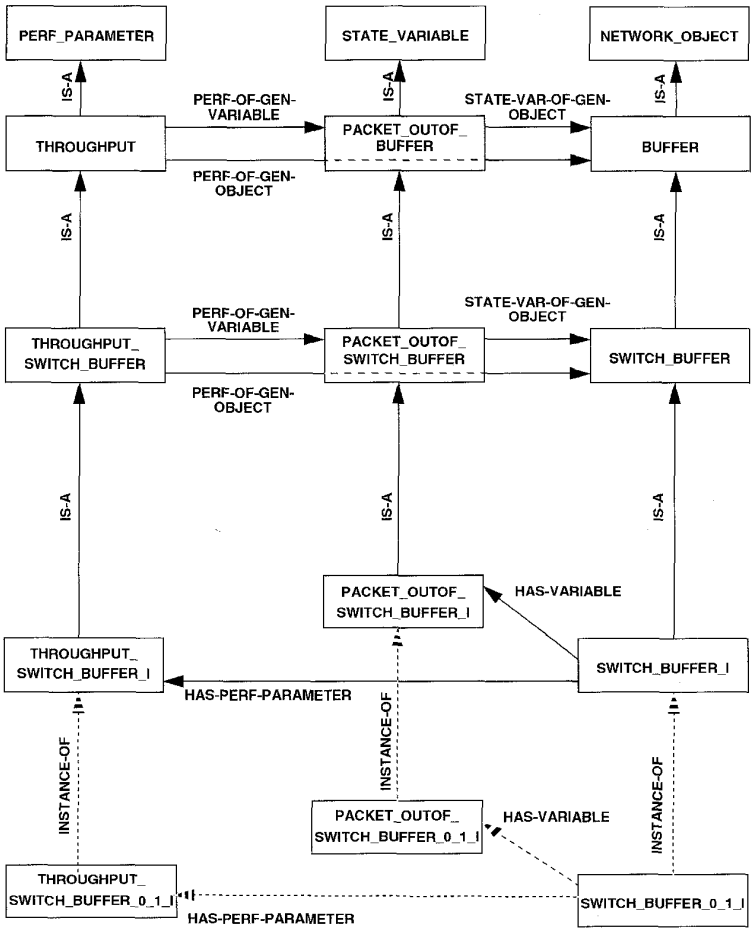

Fig. 7. Semantic network of performance parameters of BUFFER and its subclasses.

The main goal of the above algorithm is to automatcally identify the network sensors to be activated in order to compute a generic performance parameter associated with a given object class. Fig. 7. describes the semantic network associated with an instance SWITCH_BUFFER_0_1_I. In step 1, for a given object class name and and key-values, we identify the instances of the object class that matches the description. Then in step 2, we identify the instance of the performance parameters that matches the generic performance parameter specified in the query. For example, if we are interested in the THROUGHPUT of a selected instance SWITCH_BUFFER_0_1_I, then THROUGHPUT_SWITCH_BUFFER_0_1_I will be selected. Given a specific instance of a performance parameter, in step 3 we identify the corresponding instance of variable associated with the object. In Fig. 7, the variable PACKET_OUTOF_BUFFER_0_1_I is associated with THROUGHPUT_SWITCH_BUFFER_0_1_I. In step 4, we create an instance of an object class OBJ_VIEW, as shown in Fig. 8 and associate the selected performance parameters and the corresponding variables with the new object. In step 5, we create an instance of DERIVED_SENSOR and associate it with the instance of OBJ_VIEW. A side-effect of association of a derived sensor with an object is that it will create instances of primitive sensors (STATUS_SENSOR or EVENT_SENSOR) to monitor the variables associated with the object. The instances of OBJ_VIEW and DERIVED_SENSOR together form the OBSERVATION-FRAME. In step 6, instances of primitive sensor will send message to the network to activate sensors installed in the network. In step 7, statistical inferences are applied to compute the value of the performance parameter from the network measurement and sent to the observation frame in the knowledge base.

4.3 Examples

Two examples are given for objective-driven monitoring. The first shows the evaluation of the throughput of a buffer while the second gives the evaluation of the time delay of a call.

4.3.1 Monitoring a Traffic Buffer

In the following example, it is shown how to monitor the average throughput of a buffer at a network station of MAGNET II [23] network. Let us assume that the query requests the THROUGHPUT of SWITCH_BUFFER with following key attribute-value pair: **buffer_id = I** at **node_no = 0** and **st_no = 1**.

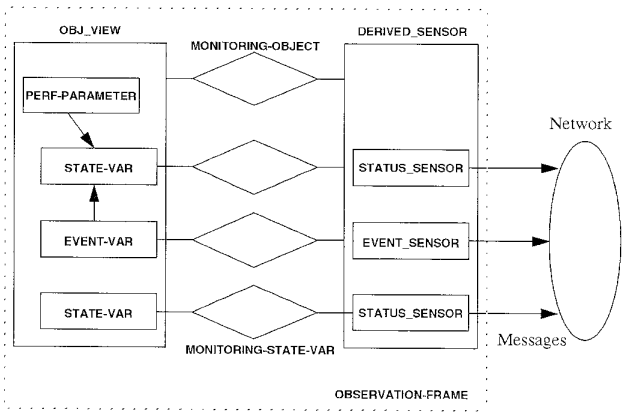

Fig. 8. The schematic of observation frame containing state variables and sensors.

Based on the specification of the attributes **node_no**, **st_no**, and **buffer_id**, SWITCH_BUFFER_0_1_I, which is an instance of SWITCH_BUFFER, is identified as the object to be monitored. Then, THROUGHPUT_SWITCH_BUFFER_0_1_I, the specific instance of THROUGHPUT for SWITCH_BUFFER_0_1_I is identified as the performance variable of the OBJECT-VIEW. Based on the relationship PERF-OF-STATE-VAR between the THROUGHPUT and PACKET_OUTOF_BUFFER, the event variable PACKET_OUTOF_BUFFER_SWITCH_BUFFER_0_1_I is identified. Once the performance parameter and the event variable are identified, an instance of the object class OBJ_VIEW is created. The new object is considered a weak entity of the SWITCH_BUFFER_0_1_I and it is uniquely identified by its own class name (OBJ_VIEW), the class name of the object being monitored and key attributes of the latter. The relationship type HAS-OBJ-VIEW-PERF-PARAMETER establishes the association between the new object and the performance parameter THROUGHPUT_SWITCH_BUFFER_0_1_I. Similarly, the relationship type HAS-OBJ-VIEW-STATE-VAR establishes the association between the new object and the state variable PACKET_OUTOF_BUFFER_SWITCH_BUFFER_0_1_I.

Once the instance of OBJ_VIEW is created, a derived sensor is attached to the object. Based on the relationship OBJECT-VIEW-OF-GENERIC-OBJECT, the derived sensor is created as an instance of SENSOR_SWITCH_BUFFER, which is a specialization of DERIVED_SENSOR for SWITCH_BUFFER. SENSOR_SWITCH_BUFFER contains the SWITCH_BUFFER specific sampling information and it is a subclass of DERIVED_SENSOR. If the object class SENSOR_SWITCH_BUFFER does not exist then the derived sensor is created as an instance of object class DERIVED_SENSOR. The relationship MONITORING-OBJECT establishes the association between the instances of OBJ_VIEW and DERIVED_SENSOR. The attributes of SENSOR_SWITCH_BUFFER are shown in Fig. 5b. The existence of the derived sensor implies the creation of an instance of EVENT_SENSOR for the event variable PACKET_OUTOF_BUFFER_SWITCH_BUFFER_0_1_I. It also establishes the relationship MONITORING-EVENT-VAR between the event variable and the instance of the EVENT_SENSOR.

Creation of the sensor causes it to send a message for activation of the primitive sensor associated with PACKET_OUTOF_BUFFER_SWITCH_BUFFER_0_1_I in the network and the activation of sensor causes start of the measurement of the variable. Once the measurement is completed and the statistical operators are applied, the value of the throughput performance parameter is sent back to the knowledge base.

4.3.2 Monitoring the Time Delay of a Call

The maximum time delay of a call appears as a QOS constraint for the Class-I traffic of MAGNET II. In order to guarantee that this requirement is met, the time-delays of Class I calls are requested.

Let CALL be the object class that represents a call with key attributes **calling_user_id**, **called_user_id**, and **traffic_class**. Let the two users of a call CALL_A_B_I be A and B and the values of attributes **calling_user_id**, **called_user_id**, **traffic_class** be A, B, and I, respectively. The association between a call and the corresponding nodes and links along the route of the call is needed. Since a node has multiple access points, the description of the route needs to include the name of the input-output buffers at all the nodes. The relationship type HAS-VCKT-ROUTE associates the buffers and servers (links and switches) along the route with the call. Thus, the route for the call between the users A and B (shown in Fig. 9), will contain the following objects: $B_{j_1,i_1,k}$, $B_{j_1,i_2,k}$, $B_{j_2,i_1,k}$, and $B_{j_2,i_2,k}$ as buffers, N_{j_1} and N_{j_1} as switch fabrics, and $L_{j_1j_2}$ as links, where the indices j_n indicate the node number, i_n indicate the access points at the node, k indicate the traffic class. The relationship type HAS-VCKT-ROUTE introduces another relationship HAS-COMPONENT-OBJECT, to reflect the fact that buffers, switches, and links form part of the call. Both relationship types HAS-VCKT-ROUTE and HAS-COMPONENT-OBJECT are included as multi-valued attributes in the list of attributes of the object class CALL.

Fig. 9. The description of the route of a call between two users.

Since the maximum time delay of a call will be the aggregation of the maximum time delays of the buffers and servers along the route of the call, an aggregate performance parameter is defined to represent the maximum time delay of the call. The new object class is called AGGR-PERF-PARAMETER. It is defined as a subclass of PERF-PARAMETER, but has an additional procedure associated with it for aggregating the values of its component performance parameters. The time delay of the object class CALL is defined as TIME_DELAY_CALL which is a subclass of AGGR-PERF-PARAMETER. The value of the **max_value** of TIME_DELAY_CALL is the sum of the values of the **max_value** of its component performance parameters.

In Fig. 10, the description of the CALL and its time delay performance parameter, TIME_DELAY_CALL, are shown. The object TIME_DELAY_CALL is declared as a subclass of AGGR_PERF_PARAMETER and associated with the generic object CALL.

```
(defschema      CALL
    (IS-A                       NETWORK-OBJECT)          ;;; inherits its attributes
    (generic_name               CALL)                    ;;; class name
    (key                        (calling_user_id called_user_id traffic_class))
    (calling_user_id)
    (called_user_id)
    (traffic_class)
    (vckt_no)
    (HAS-VCKT-ROUTE)
)

(defschema      AGGR_PERF_PARAMETER
    (IS-A                       PERF-PARAMETER)          ;;; inherits its attributes
    (generic_name               AGGR_PERF_PARAMETER)     ;;; class name
)

(defschema      TIME_DELAY_VAR
    (IS-A                       STATE_VARIABLE)          ;;; inherits its attributes
    (generic_name               TIME_DELAY_VAR)          ;;; class name
    (VAR-OF-GENERIC-OBJECT      BUFFER)                  ;;; state variable of Buffer
)
(defschema      TIME_DELAY
    (IS-A                       PERF_PARAMETER)          ;;; inherits its attributes
    (generic_name               TIME_DELAY)              ;;; class name
    (PERF-OF-GENERIC-STATE-VAR  TIME_DELAY_VAR)          ;;; relationship with state variable
)

(defschema      TIME_DELAY_CALL
    (IS-A                       AGGR_PERF_PARAMETER)     ;;; inherits its attributes
    (generic_name               TIME_DELAY_CALL)         ;;; class name
    (PERF-OF-GENERIC-OBJECT     CALL)                    ;;; relationship with object
)
```

Fig. 10. The attributes of a call object and its maximum time delay performance parameters.

In order to monitor the maximum time delay of the call, the following objective is defined:

Find the maximum time delay of a call between the pair of users A and B;

Based on the algorithm for objective driven monitoring, the steps for computation of the average time delay of a call can described as follow:

1) Create an observation frame OBJ_VIEW_CALL_A_B_I for CALL_A_B_I with:

- HAS-PERF-PARAMETER — TIME_DELAY_CALL_A_B_I
  - HAS-VCKT-ROUTE — $\text{SWITCH\_BUFFER}_{j_1,i_1,k}$, $\text{SWITCH\_BUFFER}_{j_1,i_2,k}$, $\text{SWITCH\_BUFFER}_{j_2,i_1,k}$, $\text{SWITCH\_BUFFER}_{j_2,i_2,k}$, ..., $\text{SWITCH\_BUFFER}_{j_n,i_1,k}$, $\text{SWITCH\_BUFFER}_{j_n,i_2,k}$, $\text{SWITCH\_FABRIC}_{j_1}$, ..., $\text{SWITCH\_FABRIC}_{j_n}$, $\text{LINK}_{j_1 j_2}$, ..., $\text{LINK}_{j_{n-1} j_n}$
  - HAS-OBJ-VIEW-PERF-PARAMETER — $\text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_1,i_1,I}$, $\text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_1,i_2,I}$, ..., $\text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_n,i_2,I}$
- HAS-OBJ-VIEW-STATE-VAR — $\text{TIME\_DELAY\_VAR\_SWITCH\_BUFFER}_{j_1,i_1,I}$, $\text{TIME\_DELAY\_VAR\_SWITCH\_BUFFER}_{j_1,i_2,I}$, ..., $\text{TIME\_DELAY\_VAR\_SWITCH\_BUFFER}_{j_n,i_2,I}$

2) For each $\text{TIME\_DELAY\_VAR\_SWITCH\_BUFFER}_{j_n,i_n,I}$, send a message to the corresponding sensor at the buffer to compute the maximum time delay. The location of the variable is given by the values of j_n and i_n and the buffer is identified based on the **buffer_id** (equal to I).
3) Once the value of $\text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_n,k_n}$ is computed, all the values are sent to OBJ_VIEW_CALL_A_B_I.
4) When all the $\text{TIME\_DELAY\_VAR\_SWITCH\_BUFFER}_{j_n,i_n,I}$ are available, TIME_DELAY_CALL_A_B_I is computed based on (4.1).

$$\begin{aligned}\text{TIME\_DELAY\_CALL\_A\_B\_I}.\textbf{max\_value} &= \sum_{n=1}^{R} \text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_n,i_1,I}.\textbf{max\_value} \\ &+ \sum_{n=1}^{R} \text{TIME\_DELAY\_SWITCH\_BUFFER}_{j_n,i_2,I}.\textbf{max\_value} \\ &+ \sum_{n=1}^{R} \text{SWITCH\_FABRIC}_{j_n}.\textbf{time\_delay} \\ &+ \sum_{n=1}^{R-1} \text{LINK}_{j_n,j_{n+1}}.\textbf{time\_delay},\end{aligned} \tag{4.1}$$

where R is the total number of nodes in the route of CALL_A_B and $\text{LINK}_{j_n,j_{n+1}}.\textbf{time\_delay}$ is the fixed transmission time delay of the link $\text{LINK}_{j_n,j_{n+1}}$.

5 Conclusion

A step-by-step design procedure of sensor configuration and activation for monitoring network behavior has been presented. The sensor configuration uses the modeling approach for specifying the attributes of the sensors and the procedures for sensor operations.

An objective driven measurement strategy has been presented that selectively activates the sensors needed for collecting the required information. The objectives for monitoring are obtained from the real time control task for resource management or operator submitted queries. The queries are processed by a deductive inference processor that identifies the state variables that are to be monitored. The role of the deductive inference processor is to set up an observation frame, i.e., a data space in which only data relevant to the query is allowed. The answer to queries is obtained by examining, processing, and aggregating monitored information in the data space. The sample path and statistical operators are applied to compute the performance of the network.

ACKNOWLEDGMENTS

The research reported here was supported in part by the National Science Foundation under Grant CDR-84-21402 and in part by the New York State Center for Advanced Technology under Project NYSSTF CAT (84)-15 005.

REFERENCES

[1] A. Lazar, A. Temple, and R. Gidron, "An Architecture for Integrated Networks that Guarantees Quality of Service," *Int'l J. Digital and Analog Comm. Systems*, vol. 3, no. 2, pp. 229-238, Apr.-June 1990.

[2] P.F. Pawalita, "Traffic Measurements in Data Networks, Recent Measurement Results, and Some Implications," *IEEE Trans. Computers*, vol. 29, no. 4, pp. 525-535, Apr. 1981.

[3] S. Wu and G.E. Kaiser, "Network Management with Consistently Managed Objects," *Proc. GLOBECOM '90*, pp. 304.7.2-304.7.6, San Diego, Calif., Dec. 1-3, 1990.

[4] S.L. Bernstein and J.G. Herman, "NU: A Network Monitoring, Control, and Management System," *Proc. IEEE Int'l Conf. Comm. '83*, pp. 478-479, 1983.

[5] D. Ritter and M. Seale, "A Multi-Purpose, Distributed LAN Traffic Monitoring Tool," *IEEE Network*, vol. 1, no. 3, pp. 32-39, July 1987.

[6] M. Soha, "A Distributed Approach to LAN Monitoring Using Intelligent High Performance Monitors," *IEEE Network*, vol. 1, no. 3, pp. 13-20, July 1987.

[7] P.D. Amer, "A Measurement Center for the NBS Local Area Computer Networks," *IEEE Trans. Computers*, vol. 31, no. 8, pp. 723-729, Aug. 1982.

[8] L.N. Cassel and P.D. Amer, "Management of Distributed Measurement over Interconnected Networks," *IEEE Network*," vol. 2, no. 2, pp. 50-55, Mar. 1988.

[9] A.A. Lazar, G. Pacifici, and J.S. White, "Real-Time Traffic Measurements on MAGNET II," *J. Selected Areas in Comm.*, vol. 8, no. 3, pp. 467-483, Apr. 1990.

[10] J.N. Brunken, R. Mager, and R.A. Putzke, "NEMOS—The Network Management System for the AT&T Long Distance Network," *Proc. Int'l Conf. Comm.*, pp. 99-114, Boston, June 11-14, 1989.

[11] J. LeBlanc and A.D. Robbins, "Event Driven Monitoring of Distributed Programs," *Proc. Fifth Int'l Conf. Distributed Computing Systems*, pp. 515-522, Denver, May 13-17, 1985.

[12] J. Joyce, G. Lomow, K. Slind, and B. Unger, "Monitoring Distributed Systems," ACM *Trans. Computer Systems*, vol. 5, no. 2, pp. 121-150, May 1987.

[13] P. Bates and J.C. Wilden, "An Approach to High-Level Debugging of Distributed Systems," *ACM SIGPLAN Notices*, vol. 18, no. 8, pp. 107-111, Aug. 1983.

[14] D. Wybranietz and D. Haban, "Monitoring and Performance Measuring Distributed Systems During Operation," *Proc. 1988 ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, pp. 197-206, Santa Fe, N.M., May 24-27, 1988.

[15] R. Snodgrass, "A Relational Approach to Monitoring Complex Systems," *ACM Trans. Computer Systems*, vol. 6, no. 2, pp. 157-196, May 1988.

[16] S. Mazumdar and A.A. Lazar, "Knowledge-Based Monitoring of Integrated Networks," *Integrated Network Management*, B. Meandzija and J. Westcott, eds., vol. 1. New York: North-Holland, pp. 235-243, 1989.

[17] B.P. Ziegler, *Multifacetted Modeling and Discrete Event Simulation*. London: Academic Press, 1984.

[18] S. Mazumdar and A.A. Lazar, "Monitoring of Integrated Networks for Performance Management," *Proc. SUPERCOM, '90 IEEE Int'l Conf. Comm.*, Atlanta, Apr. 15-19, 1990.

[19] S. Mazumdar and A.A. Lazar, "Objective-Driven Monitoring," *Second Int'l Symp. Integrated Network Management*, Washington, D.C., Apr. 1-5, 1991.

[20] P.P. Chen, "The Entity-Relationship Model—Toward a Unified View of Data," *ACM Trans. Database Systems*, vol. 1, no. 1, pp. 9-36, Mar. 1976.

[21] S. Mazumdar, "Knowledge Based Monitoring of Integrated Networks for Performance Management," PhD thesis, Dept. of Electrical Eng., Columbia Univ., Aug. 1990.

[22] A. Dutta, M.D. Gagle, and A.B. Whinston, "Deductive Query Processing in a Codasyl Data Base," *Proc. IEEE Workshop Languages for Automation*, pp. 200-208, New Orleans, Nov. 1-3, 1984.

[23] A.A. Lazar, R. Gidron, and A. Temple, "MAGNET II: A Metropolitan Area Network Based on Asynchronous Time Sharing," *J. Selected Areas in Comm.*, vol. SAC-8, no. 8, pp. 1,582-1,594, Nov. 1990.

Subrata Mazumdar received the PhD degree in electrical engineering from Columbia University in New York City in 1990. He has been a member of the technical staff in the Network and Service Management Research Department of Bell Laboratories since November 1995. His current research interests are in the areas of management of services in broadband networks and distributed systems.

From 1990 to 1995, Dr. Mazumdar was a research staff member with the IBM T.J. Watson Research Center, where he developed the architecture and prototype of the protocol independent network management agent and the CORBA-based OSI system management agent. He also implemented the prototype of the CORBA-based TINA-C Distributed Processing Environment for multimedia services. He was the author/co-editor of the X/Open-Network Management Forum that sponsored the specification of translation of GDMO specifications into OMG-IDL and the translation of SNMPv2 specifications into OMG-IDL.

Aurel A. Lazar has been a professor of electrical engineering and director of the Telecommunication Networks Laboratory of the Center for Telecommunications Research at Columbia University in New York since 1988. His research interests span both theoretical and experimental studies of telecommunication networks and multimedia systems. Theoretical research he conducted during the 1980s were pertinent to the modeling, analysis, and control of telecommunication networks. He formulated optimal flow and admission control problems and, by building upon the theory of point processes, derived control laws for Markovian queueing network models in a game theoretic setting.

Professor Lazar was the chief architect of two experimental networks, generically called MAGNET. This work introduced traffic classes with explicit quality of service constraints to broadband switching and led to the concepts of schedulable, admissible-load, and contract regions to real-time control of broadband networks. He is currently leading the COMET project of the Center for Telecommunications Research on the foundations of the control and management architecture of future giant gigabit networks. This research pioneered the application of virtual reality to the management of ATM-based broadband networks. He is also investigating multimedia networking architectures that support interoperable exchange mechanisms for interactive and on-demand multimedia applications with quality of service requirements.

A fellow of the IEEE, Professor Lazar is an editor of *ACM Multimedia Systems*, past area editor for network management and control of *IEEE Transactions on Communications*, a member of the editorial board of *Telecommunication Systems*, and editor of the Springer monograph series on Telecommunication Networks and Computer Systems.