An Active Temporal Model for Network Management Databases [1]

Masum Z. Hasan
zmhasan@db.toronto.edu

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A1

Abstract

The purpose of a network management system is to provide smooth functioning of a large heterogeneous network through monitoring and controlling of network behavior. ISO/OSI has defined six management functionalities that aid in overall management of a network: configuration, fault, performance, security, directory and accounting management. These management functionalities provide tools for overall graceful functioning of the network on both day-to-day and long-term basis. All of the functionalities entail dealing with huge volumes of data. So network management in a sense is management of data, like a DBMS is used to manage data. This is precisely our purpose in this paper to show that by viewing the network as a conceptual global database the six management functionalities can be performed in a declarative fashion through specification of management functionalities as data manipulation statements.

But to be able to do so we need a model that incorporates the unique properties of network management related data and functions. We propose a model of a database that combines and extends the features of *active* and *temporal* databases as a model for a network management database. This model of a network management database allows us to specify network management functions as *Event-Condition-Action* rules. The *event* in the rule is specified using our proposed *event specification language*.

1 Introduction

A network management (NM) system supporting all the six functionalities of configuration, fault, performance, accounting, security and directory management has to deal with huge volumes of data that are resident on the *management station(s)* and on the *managed entities* distributed over the network.

The system generally has to deal with two types of data: *static* and *dynamic*. Static data either never change or change very infrequently. The topology of the network, hardware and software network configurations, customers information etc. and the stored *history traces* of both dynamic and static data constitute the static portion of the NM-related data. The rapidly changing dynamic data embodies the current behavior of the network. A *Management Information Base (MIB)* defines the schema of the dynamic data to be collected for a particular

[1] Appears in Proceedings of the IFIP/IEEE Fourth Internationl Symposium on Integrated Network Management, May 1995, Santa Barbara, California, Pages 524-535.

network entity. The dynamic data distributed over the network is not visible to the network management station until they are collected. The past and present static and dynamic data form a conceptual global database which allows a management station to see the global picture of the network.

The management of a network is generally performed through two activities: *monitoring* and *controlling*. Monitoring is performed for two purposes: collection of data *traces* for current and future analysis and watching for interesting *events*. An occurrence of an event or a set of interrelated events may cause further monitoring or controlling action.

An event can be a "happening" (for example, *link down*) in the network or a pattern of data appearing in the network. The later being called a *data-pattern event* in [WSY91]. An example of a data pattern event is the crossing of a *threshold* value of a MIB variable. A data pattern event may also be defined as a more complex pattern involving more than one variables and managed entities. A set of interrelated events is called a *composite event* or *event pattern*. The interrelationship of network management events are generally *temporal*. For example, a *composite* (alert) event may be defined which occurs when the interval during which three successive server *overload* events occur is overlapped with the interval of three successive observation of large packets on the local net from unauthorized destination or the *first* crossing (up) of a rising threshold *since* the crossing (up) of a falling threshold.

Monitoring action can be performed either by asynchronous event notification (*trap*) or through periodic *polling*. Polling can be considered as an event whose occurrence at regular intervals triggers retrieval.

Both data traces and events may be stored selectively for future analysis. A *temporal database* is required for this purpose.

From the discussion above we conclude that the nature of NM data and functionalities require a model of a database that incorporates novel features of both *active* and *temporal* databases, since active databases allow one to specify events whose occurrence trigger actions and temporal databases allow one to manipulate temporal data. We propose such a model where the NM functions are specified as declarative *Event-Condition-Action* (ECA) statements. In this system, data pattern events and any other NM functions can be specified as declarative data manipulation statements. We have developed an *event specification language* (ESL) for defining composite events used in the E part of ECA. Our ESL incorporated with a temporal data manipulation language (used in the C and A part of ECA) provides us with a sophisticated declarative language for use with a database that requires active and temporal features, such as, a network management database.

The rest of the paper is organized as follows. In Section 2 we describe the features of active and temporal databases and our proposed model of a network management database. The ESL language with examples of ESL expressions and an example of an implementation of an ESL operator is discussed in Section 3. In Section 4 we provide a number of example specifications of NM functions using ECA rules. We compare our work with others in the literatures in Section 5 and conclude in Section 6.

2 Model of a Network Management Database

Before discussing our proposed model of a network management database we first discuss the features of active and temporal databases.

2.1 Active Databases

Conventional DBMSs are passive in that they manipulate data only when requests from applications are made. On the other hand, an *Active* DBMS (ADBMS) provides facilities for specifying actions or database operations to be performed automatically in response to certain events and conditions. Active behavior in an ADBMS is achieved through *Event-Condition-Action* (ECA) [MD89] rules. The rules state that when the specified event(s) occurs and the condition holds, perform the action. A condition is defined over the state of the database and its environment (for example, transaction causing the event). An action can be an arbitrary program or a database operation.

The following *primitive* events are generally supported in an ADBMS: 1) events relating to database manipulation operations, such as, retrieve, insert, delete, update; 2) transaction events; 3) absolute and relative time events; 4) in object-oriented databases method or function execution events; and 5) explicit or abstract events that are raised explicitly by the application (programmer). We also add in the list of primitive events the *data-pattern* events. A data-pattern event is specified using a database query language, for example, SQL. An event may have typed formal arguments which are bound to actual values when the event is detected. For example, the insert event may have as arguments the name of the relation and the inserted tuple.

An event is an occurrence in the database, it's environment and application's environment and can be considered as a point in time where time is modeled as a discrete sequence of points. It is desirable for many applications to react not only to current events but also to a composition or selection of events occurring at different time points. An *event algebra* allows one to specify *composite* events consisting of other primitive and composite events by means of algebra operators. A composite events expression operates on a history of events. So a composite event expression formed using algebra operators allows one to express relationship between events in the temporal dimension. The composite event happens when the specified relationship as defined by the algebra operators is detected in the event history. Petri net [GD94] or finite state machines [GJS92] can be used to model the language operators and detect composite events expressed as event expressions.

2.2 Temporal Databases

A *temporal* database in [ea93] is defined as a database that supports some aspect of time, not counting user-defined time. In other words, a TDBMS "understands" the notion of time and provides temporal operators that allow one to specify temporal queries. A temporal database contains the history of the modeled world as opposed to the traditional *snapshot* database where the past states of the database are discarded.

A temporal database contains two types of entities: *events* and *intervals*. An *event* is an instantaneous occurrence with an implicit time attribute indicating when that event occurred. Since time is generally considered as discrete, the notion of "instantaneous" requires definition. A term called *chronon* which is the shortest duration of time supported by a TDBMS, that is, a nondecomposable unit of time, is defined in [ea93]. An event occurs at any time during the chronon interval. In the network management domain we need the support for multiple choronons associated with each event entity or relation. The need for the support of multiple chronons is mentioned in [ea94]. An *interval* is the time between two events. It may be represented by a set of contiguous chronons [ea93].

2.3 Network Management Databases

Network management consists of monitoring and controlling the behavior of a network, which require the presence of sophisticated mechanism for the specification of events and correlated events occurring at different time points and specification of rules for dealing with these events. Both primitive and composite events may need to be saved in the database as events or intervals for current or future manipulation. Timestamped *trace* data which may or may not be considered as events may also need to be stored in the database. The later is called a *trace collection* in [WSY91]. The underlying datastore is thus a *temporal* database capturing the history of snapshots of network behavior. So a model of a database that combines the features of both active and temporal databases is well suited for network management databases.

The question then arises, how to specify polling, data pattern events, composite events and trace collection in a declarative way.

By considering the network as a database, the data pattern events can be specified as data manipulation statements in any declarative database language, for example, SQL. In [CH93] we specified data pattern events as GraphLog queries.

Management action is performed by monitoring on the network database. Polling or sampling is one form of monitoring. Monitoring action then consists of the following: 1) fetch the attributes specified in the *select* statement of the DML at each poll interval, 2) as data arrive, evaluate the query. If the evaluation succeeds, the data pattern event is generated. In case of trace collection, the DML statement will insert the arrived tuples in the database. The system may delegate the above functions to managed entities, if it knows that the entities can perform the functions themselves. The entities then report back the events to the manager.

This is how monitoring for a data pattern event or trace collection will be specified in our system:

```
E: poll at regular intervals
C: TRUE
A: Evaluate DML statement
```

Polling and composite events will be specified using our proposed ESL which is the subject of the next section. We specify polling in the E part as a composite event, because it is a time event occurring at regular intervals. By specifying it as a composite event using ESL we control how polling will be performed. A graphical view of the ECA mechanism is shown in Figure 1.

2.3.1 Special events

1) *poll(X)*, where X is an unique id of an ECA rule. This event may be used to start a polling action or execution of any action at regular intervals. 2) *deactivate (X)*, where X is an unique id of an ECA rule. This event may be used to *deactivate* a perpetually running instance of an event expression. Note that both *poll* and *deactivate* are events, not procedures. These events can be generated through a special function called $generate(e)$.

3 Event Specification Language

In this section we describe a language for specifying *composite* events. We define a number of *operators* which are used for composing primitive, other composite events and intervals into higher level composite events and intervals. The operators are chosen so that they are useful for

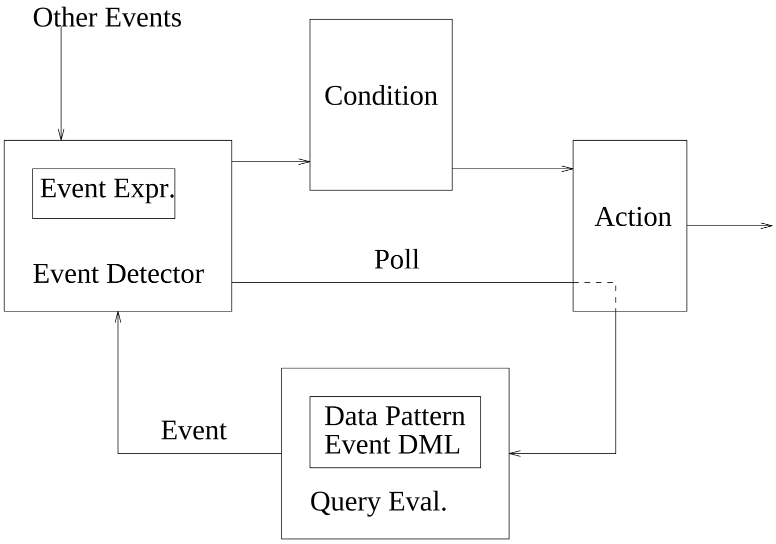

Figure 1: Graphical View of the ECA Mechanism

specifying events and intervals selections, compositions and correlations in a number of advanced application domains.

In our intended applications domain events happen in parallel in the distributed entities. It is possible to order the events totally at the central site where they are collected for processing. But this does not allow us to detect arbitrary temporal ordering, for example, overlap of intervals during which events happen. A total ordering in the event history is assumed in [GJS92]. We use Petri Net as implementation model of ESL expressions. Petri net allows reasoning about partial order of computation.

3.1 ESL Operators and Expressions

We define a number of basic operators, we think are useful for a number of applications requiring active database support. Details about the language and its implementation can be found in [Has94].

- $E = e_1 \ominus e_2$, Operator $\ominus$ defines the event that occurs when either of e_1 or e_2 occurs.
- $E = e_1 \oplus e_2$, E occurs when both of the events occur in any order.
- $E = e_1$ **fb** e_2, Event E happens when e_2 occurs any time after the occurrence of e_1.
- $E = e_1$ **se** e_2, Event E happens when e_1 occurs strictly after e_2 in the successive chronon points associated with the events.
- $E = e_1$ **in** I, E is signalled when e_1 happens in the interval I which is open at the right.
- $E = e_1$ **ne** I, E happens if e_1 does not happen in the interval I, E is signalled at the end point I_e of I.
- $E =$ **first**(e), This operator selects the first e event from a series of *consecutive* or *concurrent* e events in the event history.

- E = **last**(e), Same as *first*, but the last event is selected.
- An interval between two events e_1 and e_2 is specified as $[e_1, e_2]$.

We will now provide a number of useful additional operators.

- e_3 **fs** e_1 = **last**(e_3) **fb** **first**(e_1), specifies first e_1 event since (after) the recent e_3.

In the network management domain *persistence* of an event in an interval may be of interest. Since the model of time is discrete, rather than continuous, persistence has to be defined in terms of the discrete model of time. If an event happens at *all* chronon points associated with the event in the specified interval, then that event is said to persist for that interval.

- e_1 **pe** I = (...((e_1 **se** e_1) **se** e_1)...) **se** e_1) **in** I, defines the *persistence* of an event, which happens when e_1 events happen in strict sequence at each chronon point in the interval I.

3.2 Implementation Model of ESL Operators

In this section we will provide an implementation model of the ESL operators using *colored Petri net* (CPN).

A CPN is a directed graph with two kinds of nodes, *places P* and *transitions T*, interconnected by *arcs A*. Arcs may be inscribed with *arc expressions* and transitions with *guard expressions*. A *colored token* of a CPN, as opposed to simple Petri net, can carry complex information. Places are depicted as circles and transitions as vertical line segments.

The behavior of a CPN is described as follows. A transition fires, when it is *enabled*. A transition is enabled when the variables of *input* arc expressions can be bound with appropriate tokens or colors present on the input places and evaluated, and the guard (if present) evaluates to true. When a transition fires, tokens are removed from the input places and placed on the output places. The number of removed/added tokens and the colors of these tokens are determined by the value/type of the corresponding input and output arc expressions evaluated with respect to bindings in question.

Figure 2 shows the CPN implementation of e_2 **fs** e_1. The upper portion of the figure corresponds to **last**($\mathbf{e_2}$) before the first e_1 appears. Since the last e_2 token is removed from p_1 when t_3 fires, all e_1s appearing after the firing and until the occurrence of next e_2, will be removed. A is an *auxiliary* place which is *marked* initially. Any e_1s appearing before e_2 will be removed. If both t_1 and t_2 are enabled concurrently, then we resolve the firing sequence in favor of the *terminator* event e_1, that is, t_2 will fire first, thus removing the e_1 event.

3.3 Example ESL expressions for NM

We will now give a number of examples showing how the above operators can be used for declaratively specifying interesting events of interest in the network management domain.

- A *server_underutilized (su)* event follows a router *congestion (co)* event within 2 minutes.

$$\boxed{(co \ \mathbf{fb} \ su) \ \mathbf{in} \ [co, (2 \ minute)]}$$

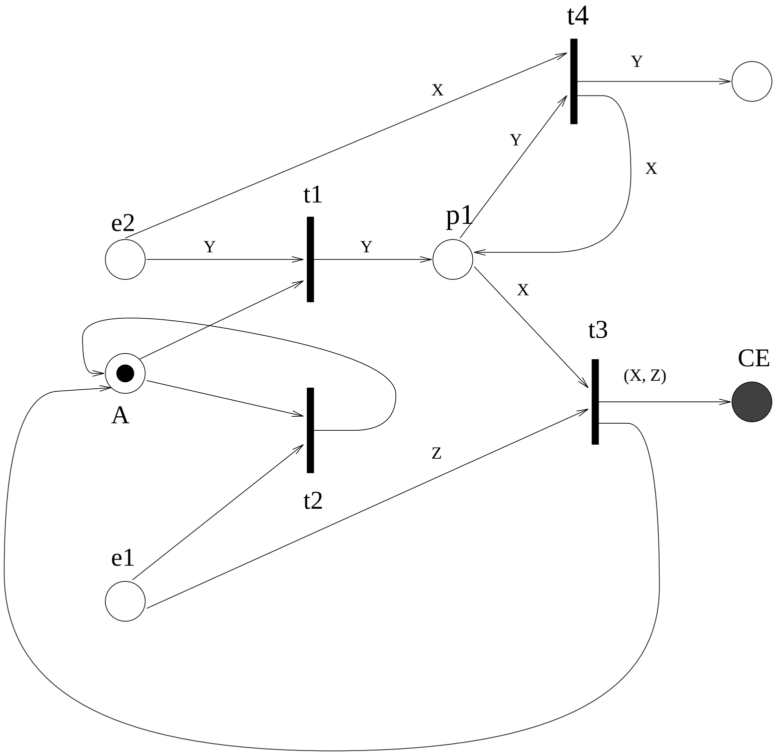

Figure 2: Petri Net Model of e_2 **fs** e_1

- *Polling* or *Sampling* is an important function in network management.

  An event of polling every 2 minutes for 1 hour can be specified as follows:

  $$\boxed{CE_3 = (2\ minute)\ \mathbf{in}\ [\mathbf{last}(poll(X), (60\ minute)]}$$

  The timer is started when the (recent) *poll* event is detected. The expression is then used to control the duration of the timer that emits (time) events every 2 minutes.

  In some cases, polling may be stopped when requested explicitly. Following expression CE_4 polls every two minutes in an interval delimited by the *poll* and *deactivate* events.

  $$\boxed{CE_4 = (2\ minute)\ \mathbf{in}\ [poll(X), deactivate(X)]}$$

- If the expression "$value \geq threshold$" is contained in the definition of an event, then the event will be generated at each sampling interval as long the value remains high. An ECA rule using this event will fire the action repeatedly which may be undesirable. What we need is some filtering mechanism to prevent this. For example, *first event since some other event* or the *hysteresis mechanism* as defined in the RMON specification [Wal]. The mechanism by which small fluctuations are prevented from causing alarm is referred to in the RMON specification as *hysteresis* mechanism.

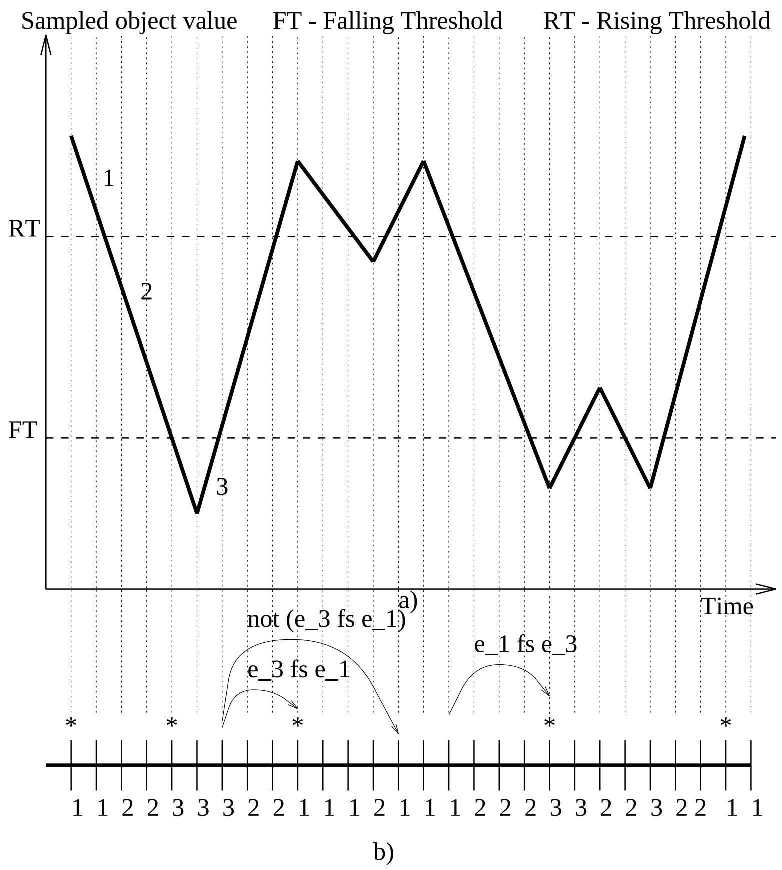

Figure 3: Specification of Hysteresis Mechanism

Hysteresis mechanism is best explained through the Figure 3.a (similar to the figure in [Sta93], we modify it to suit our purpose). As the rules for the hysteresis mechanism stipulates only the events marked as stars (*) will be reported. We assume that the events are reported at each sampling interval. Then the hysteresis mechanism can be specified as follows.

$$\boxed{CE_6 = (e_1 \textbf{ fs } e_3) \ominus (e_3 \textbf{ fs } e_1)}$$

A large number of interesting event patterns can be specified using ESL as opposed to programming or hardcoding limited set of rules in the system (like the hysteresis mechanism only in RMON). For example, if we consider Figure 3, events (such as, *server_overload*) in the region 1 may persist for long time. But that persistence event will not be generated by the hysteresis mechanism, thus leaving no room for taking action to alleviate the problem.

4 Example ECA Specifications

We now provide a number of example specifications of NM functions employing ESL, active and temporal databases concepts in an unified framework.

The SQL query Q1 in the rule RL1 below defines a *server_underutilized* (S_U) data pattern event.

```
RL1:
        E: CE4
        C: TRUE
        A: Q1
Q1:
GENERATE S_U (HOST, TCPINSEGS) AS
        SELECT HOST, TCPINSEGS
        FROM MIB_TCP
        WHERE HOST_TYPE = 'server'
                AND (TCPINSEGS - PREVIOUS(TCPINSEGS))
                        ≤ falling_threshold
```

Note that, Q1 refers to both static configuration data (topology information) and dynamic MIB data of managed entities. The implementation will evaluate the query over the configuration database once and filter out the servers. The servers will then be polled for *tcpInSegs* MIB variable values and as data arrive the crossing of threshold value will be checked. We assume that the underlying temporal database supports a temporal operator called *previous* which returns the last reported tuple (fetched in the previous poll). ECA rule RL1 specifies that the MIB_TCP tables are polled every two minutes until a *deactivate* event happens. Event expression CE_4 discussed in the previous section will serve the purpose. We assume that *poll(RL1)* event is generated initially.

Q1 can be specified as a *trace collection* which collects the traces in a table. Rule RL2 defines this trace collection.

```
RL2:
        E: CE4
        C: TRUE
        A: Q2
Q2:
INSERT INTO SERV_TCP_TRACE (HOST, TCPINSEGS)
        SELECT HOST, TCPINSEGS
        FROM MIB_TCP
        WHERE HOST_TYPE = 'server'
```

The following rule RL3 then specifies the generation of the S_U events. The *insert* is a database manipulation event.

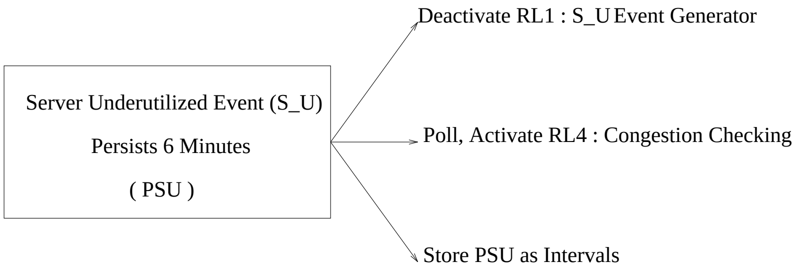

Figure 4: Diagramatic View of RL5

```
RL3:
        E: insert (SERV_TCP_TRACE, HOST, TCPINSEGS)
        C: (TCPINSEGS - PREVIOUS(TCPINSEGS))
                ≤ falling_threshold
        A: generate (S_U (HOST, TCPINSEGS))
```

We will now write an ECA rule (RL5) for the specification of the following. Watch for the *persistence* of S_U events for, say, 6 minutes. If it persists, then check for *congestion* on the routers that are on the way between the server and its clients. To detect congestion start evaluating for 1 hour every 2 minutes the corresponding data pattern event query (the corresponding rule RL4 is not shown for brevity). Deactivate the generation of S_U events and store the persistence of S_U events (PSU) as intervals in the database. A diagramatic view of RL5 is shown in Figure 4.

```
RL5:
        E: PSU (int (Self), H, V) = S_U(H, V) pe (6 minute)
        C: TRUE
        A: Q5 AND
           generate (poll (RL4)) AND
           generate (deactivate (RL1)) AND
           INSERT INTO SERV_UNDUTIL_PERSIST PSU
```

Query Q5 filters out the routers between the server and its clients. We do not show query Q5 here. Similar query can be found in [CH93]. The routers found are passed to the query portion of RL4. PSU is defined as an interval. The interval is calculated using the *int* operator on the persistent composite event PSU. Operator *int* returns the timestamps of the end points of an interval.

5 Related Work

The database issues for network management similar to the ones discussed in this paper have also been considered in [WSY91]. We provide a more uniform and consistent framework for specifying *data pattern events* and *trace collections*, that is, as ECA rules. They provide a separate mechanism for specifying trace collections. The main difference with our work is in our proposed composite event specification language, ESL. Their work lacks such an event specification language. As a result, polling and other composite events can not be specified in their system, that could control uniformly the collection of data pattern events, traces and other actions, as is done in our system. We also provide a consistent mechanism to collect events and traces in a *temporal database*. The notion of *persistence* is mentioned in their work, but no formal definition of it is provided. The MANDATE MIB project [HBNRD93] also addresses similar network management database issues. But the proposal for a unified framework for incorporating active and temporal databases concepts in a network management database similar to ours is lacking in their work. The work in [Shv93] discusses only the issues of a *static* (historical) temporal database for network management data.

6 Conclusion

We have proposed a model for network management database where the network management functions are specified as *Event-Condition- Action* rules. In proposing the model we have considered unique properties of NM data and functionalities. We have designed a temporal event and interval specification language that allows us to specify composite or (temporally) interrelated events.

Work is in progress to implement efficiently the ESL operators. Visual specification of ESL expressions and visualization of event detection process will be helpful in many application domain, including network management. We are working towards that goal. As a future work we plan to incorporate real-time or hard-deadline issues in the language.

Acknowledgments

I would like to thank Prof. Alberto Mendelzon of University of Toronto for his fruitful suggestions and support. I also thank Prof. William Cowan of University of Waterloo for his support. I specially thank Michael Sam Chee of Bell Northern Research, Ottawa, Canada for his many suggestions.

The work was supported by The Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.

References

[CH93] Mariano Consens and Masum Hasan. Supporting network management through declaratively specified data visualizations. In H.G. Hegering and Y. Yemini, editors, *Proceedings of the IEEE/IFIP Third International Symposium on Integrated Network Management, III*, pages 725–738. Elsevier North Holland, April 1993.

[ea93] C. Jensen et. al. Proposed temporal database concepts - may 1993. In *Proceedings of the International Workshop On an Infrastructure for Temporal Databases*, pages A–1–A–29, June 1993.

[ea94] N. Pissinou et. al. Towards an infrastructure for temporal databases, report of an invitational ARPA/NSF workshop. Technical Report TR 94-01, Department of Computer Science, University of Arizona, March 1994.

[GD94] S. Gatziu and K. Dittrich. Detecting composite events in active database systems using petri nets. In *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering*, pages 2–9, February 1994.

[GJS92] N. Gehani, H. Jagadish, and O. Shmueli. Composite event specification in active databases: Model and implementation. In *Proceedings of the 18th International Conference on Very Large Data Bases*, 1992.

[Has94] Masum Z. Hasan. Active and temporal issues in dynamic databases. PhD Thesis Proposal, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1994.

[HBNRD93] J. Haritsa, M. Ball, J. Baras N. Roussopoulas, and A. Datta. Design of the MANDATE MIB. In H.G. Hegering and Y. Yemini, editors, *Proceedings of the IEEE/IFIP Third International Symposium on Integrated Network Management, III*, pages 85–96. Elsevier North Holland, April 1993.

[MD89] D. McCarthy and U. Dayal. The architecture of an active data base management system. In *Proceedings of the ACM-SIGMOD 1989 International Conference on Management of Data*, pages 215–224, 1989.

[Shv93] A. A. Shvartsman. An historical object base in an enterprise management director. In H.G. Hegering and Y. Yemini, editors, *Proceedings of the IEEE/IFIP Third International Symposium on Integrated Network Management, III*, pages 123–134. Elsevier North Holland, April 1993.

[Sta93] W. Stallings. *SNMP, SNMPv2, and CMIP, The Practical Guide to Network Management Standards*. Addison-Wesley Publishing Company, Inc., 1993.

[Wal] S. Waldbusser. Remote network monitoring management information base. RFC 1271, Carnegie Mellon University.

[WSY91] O. Wolfson, S. Sengupta, and Y. Yemini. Managing communication networks by monitoring databases. *IEEE Transactions on Software Engineering*, 17(9):944–953, September 1991.

About the Author

Masum Z. Hasan is a Research Associate at the Computer Systems Research Institute, University of Toronto and a Ph.D candidate in the Department of Computer Science, University of Waterloo. He obtained BEng, MEng in Computer Engineering from former USSR and MMath in Computer Science from University of Waterloo. His research interests are in active temporal databases, network management, networked document browsing/searching, distributed and parallel programming environment, visualization. Mr. Hasan has worked for industry both in Bangladesh and Canada.