In *Integrated Network Management, V* (A. Lazar, R. Saracco, and R. Stadler (eds.)), pp. 752-765, London: Chapman & Hall Publishers, 1997.

A Spreadsheet-Based Scripting Environment for SNMP *

Pramod Kalyanasundaram, Adarshpal S. Sethi, Christopher M. Sherwin, and Dong Zhu (email: {kalyanas, sethi, sherwin, dzhu}@cis.udel.edu)

Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716

Abstract

The existing SNMP management framework does not effectively support a hierarchical management strategy. Further, existing MIBs have a static structure and do not permit dynamic user organization of management information. This paper presents a spreadsheet paradigm that allows users to dynamically configure management information and set up control at an intermediate manager. This paradigm augments the basic SNMP framework by providing value added functionality at a proxy node so that it can function as an intermediate manager. The design of a proxy MIB, a scripting language, and event model that form an integral part of the paradigm are presented with future research directions.

Keywords: Spreadsheet Paradigm, Scripting Language, Event Model, SNMP, Proxy Agent, Intermediate Manager

1 INTRODUCTION

A hierarchical management strategy is an effective means of managing the large and complex internetworks that are in use today. However, the most popular management framework, the SNMP framework (which includes both the SNMP and the SNMPv2 protocols) [RM90, CFSD90, RM91, CMRW96], is largely used in a flat model with a single manager communicating with a large number of agents. The SNMP framework defines the concept of a *proxy agent* as an agent that acts on behalf of other agents. But traditionally, SNMP has used proxy agents in a *pass-through* role, wherein a proxy might facilitate the implementation of administrative or security policies but otherwise passes the manager requests and agent responses through in an essentially transparent mode. The fact that a proxy can be used as an intermediate manager for hierarchical management is recognized by SNMPv2, and the protocol even includes an Inform Request PDU intended for manager-to-manager communication. However, it cannot be effectively used because the framework lacks essential support for hierarchical management; it provides no means for managers to delegate tasks to intermediate managers or to communicate with the intermediate managers during the execution of these tasks.

Management by delegation is a well-known strategy [YGY91, Gol95, GY95] for implementing hierarchical management, but so far the SNMP community has been unable to take advantage of it because the delegation primitives have not been integrated with the SNMP framework. In this paper, we present a new paradigm – which we call the *spreadsheet paradigm* [SK94, KSS96] – that incorporates management by delegation concepts into the SNMP framework to facilitate hierarchical management.

*This work was supported in part by the U.S. Department of the Army, Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0002 Federated Laboratory ATIRP Consortium.

The main objectives of the spreadsheet paradigm are: 1) to introduce a powerful intermediate manager that enhances (but preserves) the existing SNMP framework, provides value added functions, and supports delegation. 2) to provide an environment that supports user configurability of management information independent of the underlying MIB structure. 3) to support basic primitives, events and operations via a scripting MIB and language that allow a user to build fairly complex network management tasks. 4) to present the user with an abstraction and interface that is easy to comprehend and use.

The two essential features supported by this paradigm are: 1) specification of dynamic relationships between objects *across MIBs* and 2) flexible, hierarchical event building. Relationship specification allows two or more objects belonging to different MIBs to be related by a logical or temporal conditions. Hierarchical event building allows simple events (in cells) to be used to build more complex events.

Existing models [CMRW96] allow only event definitions and object relationships that are predefined in the MIBs supported by an agent. RMON [Wal95] allows users to set up monitoring information including event and threshold specification. However, the events that can be specified are fairly restrictive. Moreover, the user cannot set up operations to be performed on management information across nodes. Also, the user cannot set up control functions that can be executed. To allow managers to perform more sophisticated control functions, there is a need for a paradigm that permits *dynamic* specification of object relationships and event definitions. The spreadsheet paradigm facilitates dynamic configuration of information and control by providing the ability to selectively structure management information into views [†]. Customizable structuring of management information is a key feature needed by network managers that is missing in the SNMP framework.

This research uses the concept of management by delegation (MBD)[YGY91] but differs from the conventional MBD in that the delegation aspects are incorporated into the existing framework and hence will conform to the security aspects defined by the framework. Research has been done to provide the user with different views of MIBs [AY95]. These extensions make use of additions to the structure of management information (SMI) and special compilers to compile the MIBs. However, these extensions provide the user with just table operations as would be permitted by a database engine and the user cannot build custom views that are specific to a user's environment. Studies of temporal and event models have been done by several researchers, [Has95, CJS93] etc. Our work uses the concepts identified by these research directions in temporal and event models, and adapts these results to suit the spreadsheet paradigm.

This paper is organized as follows: Section 2 introduces the spreadsheet paradigm; Section 3 presents the Spreadsheet Management Information Base (MIB) design ; Section 4 covers the Spreadsheet Scripting Language (SSL) features; Section 5 describes the event model supported by the spreadsheet paradigm; Section 6 includes an example and Section 7 summarizes the conclusions and outlines future directions.

[†]The term *view* has a different meaning in SNMP terminology. We use this term to indicate user structured management information set up in the spreadsheet.

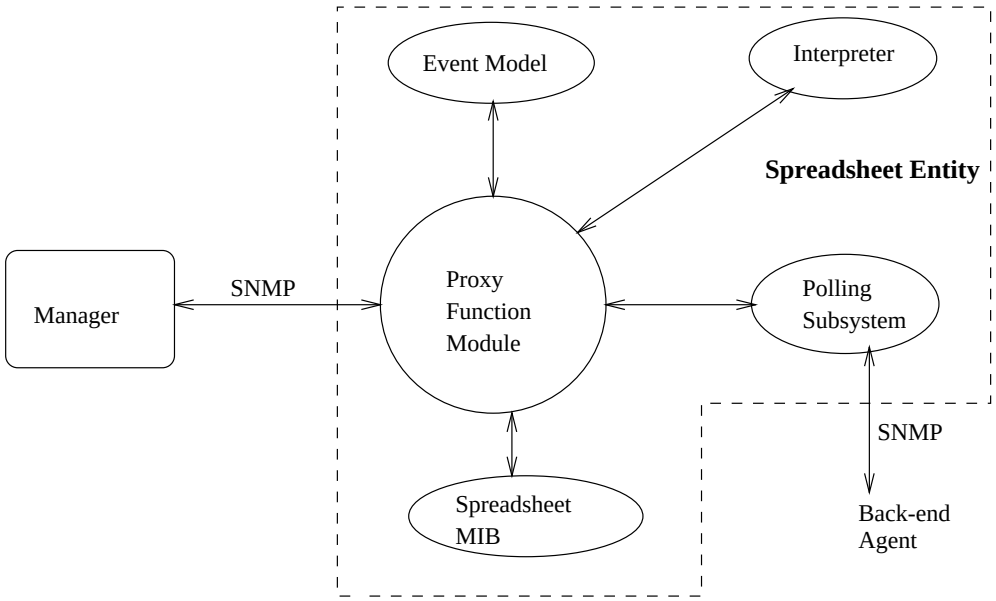

Figure 1 Spreadsheet Entity

2 THE SPREADSHEET PARADIGM

An essential component of the spreadsheet paradigm is an abstraction of a spreadsheet. A spreadsheet is composed of *cells* arranged in a two-dimensional matrix consisting of *rows* and *columns*. A cell is the fundamental unit of operation in the spreadsheet paradigm. A cell contains a *control information part* and a *data part*. The control part dictates the rules for collection of information or relationships between objects. The data part contains the data collected as a result of executing the control information specification. For instance, the control part may specify that the cell should contain the result of summing two or more counters in different nodes (or different MIBs). The data part contains the result of such a summation. More details on the operation of the spreadsheet paradigm and its various scenarios are available in [SK94, KSS96].

Figure 1 shows the basic components contained in an implementation of the spreadsheet entity located at a proxy (or an intermediate manager). The proxy accepts SNMP requests from the manager and enters them into the spreadsheet that is maintained at the proxy. The control information entered into the spreadsheet contains scripts written in the Spreadsheet Language (SSL), a specially designed scripting language for use with the spreadsheet paradigm. The scripts are interpreted locally at the proxy which may result in SNMP requests to be forwarded to one or more agents. The agents' responses are processed as specified by the scripts in the spreadsheet cells. As a result of this processing, updates may occur in the data contained in one or more cells.

The spreadsheet MIB (*ssmib*) implements the spreadsheet abstraction using SNMP tables. User operations on cells map to operations on tables that are part of this MIB. The proxy function module coordinates the activities of the various components at the proxy node. When the proxy receives an SNMP request from the manager, the proxy function module performs the necessary operations on the spreadsheet MIB to implement the cell abstraction. Once the request has been carried out, the proxy function module

responds to the manager that requested the operation. If a control value is entered into a cell, the objects that need to be polled are forwarded to the polling subsystem. The proxy module interacts with the event model to perform event based processing of the spreadsheet. When the script in a cell needs to be executed, the proxy function module interacts with the interpreter to process the cell control information.

When a user sets up information in a cell, there is a need to constantly update the value(s) contained in the cell. This can be achieved by polling the managed objects referenced in the cell. Thus, the spreadsheets cells at the proxy reflect the current values of the managed objects in the cells (within a certain time granularity). The polling subsystem supports cell-based polling entry creation, deletion and retrieval.

The polling subsystem optimizes the number of polls issued to the back-end agents by grouping variables based on: 1) time intervals and 2) hosts. The polling subsystem collects the variables that need to be polled at a single agent and issues the minimum number of poll requests to satisfy the polling specification for the variables. Also, if a single variable at a given agent is to be polled at different frequencies, the polling subsystem computes that minimum frequency of polling that will satisfy all the polling requests for the variable.

The spreadsheet paradigm supports a scripting language that can be used by a user to set up sheets of control. This language is described in Section 4. The spreadsheet language is interpreted by an interpreter and scripts that are set up in the various cells can be executed under the control of this interpreter. The interpreter performs the functions of syntax checking, run time error checking, detection and reporting.

The spreadsheet supports both the request/response (synchronous) and event (asynchronous) modes of operation. In the synchronous mode, the manager requests some operation to be performed using one of the standard SNMP protocol operations and the proxy responds after processing the request. In the asynchronous mode, the user sets up events to be watched, and actions associated with such events. On occurrence of any of the watched events, the proxy carries out the associated actions which may include notifying the manager. This mode of operation allows the manager to successfully delegate some of its routine tasks to the proxy.

3 MIB DESIGN

The proxy based *ssmib* captures the control and data part of a cell using two tables: 1) control table and 2) data table. The structure of the MIB is shown in Figure 2. The control table (*controlTable*) is made up of a sequence of *controlEntry* elements. The *controlEntry* contains the column variables that constitute a row in the *controlTable*. The variables that form the control table index are: 1) the primary internet address of the manager who is the owner of the row 2) the spreadsheet number 3) the column id of the cell and 4) the row id of the cell. The *controlEntry* also contains a *controlstring* field that holds the control information of the cell.

Similarly, the *dataTable* is made up of a sequence of *dataEntry* elements. The *dataEntry* contains the column variables that form a row in the *dataTable*. The *dataTable* index variables are the same as those of the *controlTable* with one difference – there is an additional index variable *dataValueIndex*. This additional index variable helps to uniquely identify a particular data value in a cell that contains multiple data values.

The relationship of the control and data tables with a cell is shown in Figure 3. The

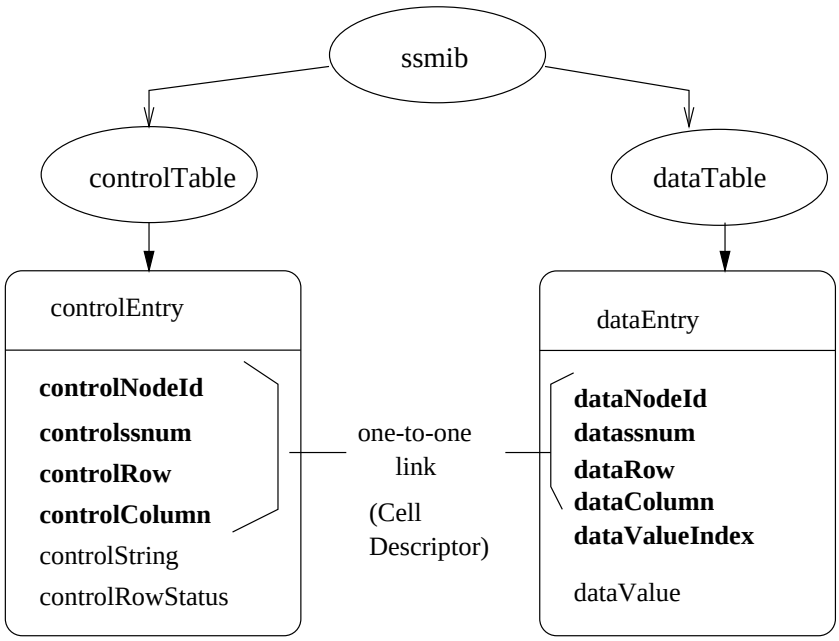

Figure 2 Spreadsheet MIB

common variables (NodeId, ssnum, Row and Column) establish a one-to-one correspondence between the control and data tables and will be referred to as the *cell descriptor*. The cell descriptor variables uniquely identify a cell at a proxy node.

In order to create a cell, the manager creates a row in the control table. To set the control portion of the cell, the manager sets the *controlstring* variable in the *controlEntry*. Depending upon the number of values defined by the control portion of the cell, an appropriate number of rows are created in the data table with the common cell descriptor values. The cell abstraction is thus represented as a row in the *controlTable* (the control part) and one or more rows in the *dataTable* (the data part) of the cell. The manager can delete a row in the control table, and based on the control to data table association, the appropriate rows in the data table are removed. To modify the control part of a cell, the manager performs a **Set** on the specified cell which translates to a **Set** on the appropriate row in the control table. If the control portion needs to be retrieved, this operation translates to a retrieval of a row from the control table. The same operations and translations apply to the data portion of the cell. The only difference arises when there are multiple values in the data portion of the cell. In such a case, the MIB design permits the retrieval of one or more values using the standard **GetBulk** operator. This is achievable since all the values in a cell have the same cell descriptor as a prefix.

Although multiple managers can set up one or more spreadsheets at the proxy, each spreadsheet is controlled by one and only one manager. This may cause some cells to be duplicated. However, duplication of cells does not affect the polling since the polling subsystem computes an optimized polling stream.

Consider the following example. A manager (node id m) wants to set up a cell (with row id i and column id j in a spreadsheet s such that it contains the same managed object on n different nodes. This results in the manager forwarding a set request to create a row in the control table. The OID of the cell will be the OID of the control table suffixed by

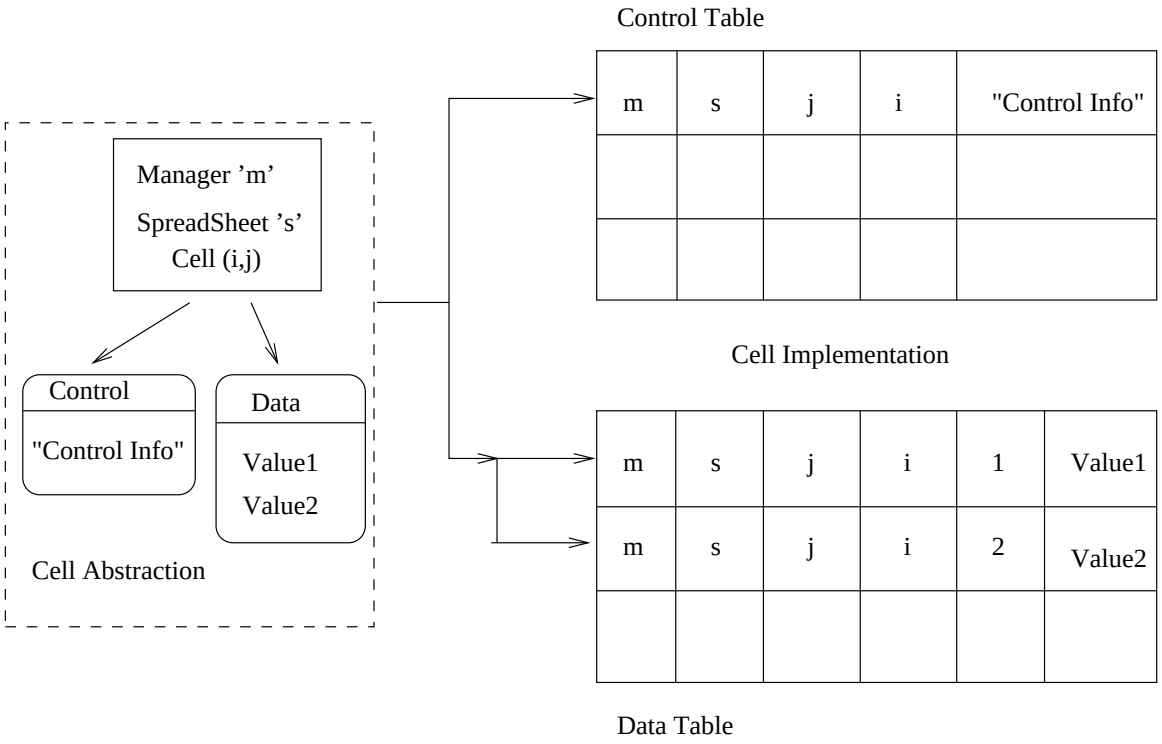

Figure 3 Cell abstraction and implementation

m.s.j.i. The cell column id precedes the cell row id since this will be consistent with the way the SNMP **GetNext** operator works. If a *GetNext* is performed repeatedly, starting at a given cell, then all the cells in the first column of the spreadsheet will be returned before the cells in the second column of the spreadsheet are returned. Once the row is created, the manager can issue a set request to set up the control portion of the cell by setting the *controlstring* variable of the row with index *m.s.j.i.*. As part of the control information set up, for each variable encountered in the control variable, a unique row is created in the dataTable with the index *m.s.j.i.v* where v takes values 1 through n (since there are n managed objects).

Deleting a cell results in the n data rows (*m.s.j.i.1* through *m.s.j.i.n*) being deleted, followed by the control row with index *m.s.j.i* being deleted. The semantics for setting the control part of the cell can be defined as a delete followed by a create or deleting the data rows associated with the old control information, changing the control information and creating new data rows that correspond to the new control information. In order to retrieve the control portion of the cell, the manager issues a get request to the proxy with the OID of the control table suffixed with index *m.s.j.i*. In order to retrieve the data portion of the cell, the manager can perform one of two operations:

1) perform a sequence of n **GetNext** operations that will return the n values contained in the cell, or 2) perform a **GetBulk** with the OID of the *dataTable* suffixed with the cell descriptor *m.s.j.i* and specify a value of n for the repeaters.

A key aspect of the MIB design has been demonstrated in the above example. Although each of the n variables contained in the cell belongs to different nodes, the user (or manager) can retrieve the values in an order that is required by the user and which has been set up by the user. The user can thus configure the management information

selectively and view it in an order that is different from that of the underlying MIB. Such reordering of management information will allow users to set up summaries and different views of the management information. Thus, a user can view only the information that is needed and in an order that conforms to the user's current needs. This dynamic configuration of the MIB does not exist in the current SNMP management framework.

4 SPREADSHEET LANGUAGE (SSL)

A language that targets a network management environment must be able to support features that facilitate the specification of network management tasks coupled with user flexibility and expressive power [Hol89]. This section describes the main features of the spreadsheet language (SSL) that forms an integral part of the spreadsheet paradigm and supports the development of network management scripts.

SSL supports features that can be broadly classified as: 1) standard procedural language features and 2) spreadsheet paradigm specific features. The standard procedural language features include: operators (arithmetic, logical and relational), control flow constructs, expression evaluation and assignment, and local variable support within a cell. The paradigm-specific features include: cell access, managed object and polling specification, multiple values in cells (or customizable views), and event specification.

SSL supports standard arithmetic operators like '+', '-', '/', '*' and relational operators like greater than ('>'), less than ('<'), and not equal to ('<>'). Cells and numbers can be used in arithmetic and relational operations. SSL supports logical-and (&&), logical-or (||) and logical-not (!) operators. All these operators allow the user to specify relationships between any set of arbitrary objects within the management domain.

Control flow constructs like **if...else...endif** and **while** allow a user to set up conditional and iterative scripts in cells. The semantics of these constructs are similar to that of a standard procedural language. However, iteration has an implicit, user-configurable, maximum loop count that ensures that a single script will not run forever.

The SSL permits a user to specify a fully qualified managed object in both symbolic and dotted decimal format. A fully qualified managed object is a combination of both the OID of the managed object and the host on which it resides. For instance, if a user wanted to monitor *tcpActiveOpens* on host *stimpy*, the user could do this by specifying *tcpActiveOpens@stimpy*. This feature allows the user to identify and access any managed object in the management domain. A user can optionally specify a polling interval for a managed object. This option allows the variable to be polled at the user-specified frequency instead of a default frequency.

The language is tightly coupled to the spreadsheet paradigm and offers facilities to access and manipulate cells. A user can specify cells using a [spreadsheet:row:col] specification (assuming a default manager). Thus, [1:2:3] would refer to a cell that occupies the second row and third column in the first spreadsheet. The language also supports assignments to cells. This allows cells to be cleared and copied. In addition, the SSL allows cells to be named (labeled) and the symbolic names to be used in scripts. Each cell supports a set of special local variables for temporary storage during script processing.

An important feature supported by the SSL is that it allows a user to set up and access multiple managed objects in a cell. A user can specify a list of objects to indicate that multiple values must be stored in the cell. Each individual value can be accessed

using [s:r:c].n where n represents the *nth* value in the cell. For example, a user can set up instances of a single counter (e.g. an error counter) from different nodes in a management domain within a single cell. When the manager retrieves the contents of this cell, a summary of the error counters on the nodes of interest can be obtained.

The SSL allows dynamic configuration of a spreadsheet using the *activate* and *deactivate* statements. Using these statements, a user can enable or disable the script contained in a cell. For example, if a set of counters need to be collected for a specified period of time until a fault occurs, a user could down-load the script that collects the counters but can deactivate it on occurrence of the fault. The *activate* statement allows the user to activate the collection script later. The script contained in a cell that is deactivated cannot be executed unless it is activated again. A cell cannot be activated or deactivated when the script in the cell is executing.

The SSL combines standard procedural language constructs with event specification constructs to provide a simple yet powerful platform for developing scripts that perform network management tasks. The event model is described in detail in Section 5. The following simple example demonstrates some useful features of the spreadsheet language and an important concept of the information model of the spreadsheet paradigm, which is the MIB view. The following example cell is set up to poll 8 counters from two different MIB-II groups on two different hosts and sum the information.

The polled values and summation information are all stored in a single cell as multiple values. The lexicographical order of these values are $$.1, $$.2, upto $$.12, where $$ is a special variable that represents the cell's value. In this way the cell presents to the manager a different MIB view which includes non-contiguous managed objects from different MIB groups, and even across different hosts. With cells properly set up to utilize this feature, significant power, flexiblity and efficiency could be achieved by the management application.

```
Cell [1,1]:
    label: cellMibViewExample
    action:
    {
        // Poll counters at sol
        $$.1 = tcpActiveOpens@sol.cis.udel.edu;     // TCP group counters
        $$.2 = tcpPassiveOpens@sol.cis.udel.edu;
        $$.3 = ipInReceives@sol.cis.udel.edu;       // IP group counters
        $$.4 = ipInHdrErrors@sol.cis.udel.edu;

        // Poll counters at tweety
        $$.5 = tcpActiveOpens@tweety.cis.udel.edu;  // TCP group counters
        $$.6 = tcpPassiveOpens@tweety.cis.udel.edu;
        $$.7 = ipInReceives@tweety.cis.udel.edu;    // IP group counters
        $$.8 = ipInHdrErrors@tweety.cis.udel.edu;

        // Summing:
        $$.9  = $$.1 + $$.2;  // Total TCP connections at sol
        $$.10 = $$.5 + $$.6;  // Total TCP connections at tweety
        $$.11 = $$.3 - $$.4;  // Total IP datagrams 0 with
                              // no header errors at sol
```

```
    $$.12 = $$.7 - $$.8;  // Total IP datagrams recveived with
                          // no header errors at tweety
}
```

5 OVERVIEW OF THE EVENT MODEL

The SNMP framework is predominantly synchronous. The primary source of asynchronous processing is the use of traps from the agent to the manager. This section describes an event model that provides asynchronous processing support for the spreadsheet paradigm and enhances the value-added capabilities provided by the proxy.

Events form the basis for the event model. An *event* is an occurrence that causes: 1) a change in the control or data part of one or more cells 2) a system related change (e.g., a timer tick, SNMP PDU receive or send) or 3) the execution of one or more cells in a spreadsheet.

Events can be either *basic* or *user-defined. Basic events* are intrinsic to the event model and are either SNMP or system related. These events form the basic building blocks for an event hierarchy. *User-defined (or derived)* events are those events that are built using a combination of basic and other user-defined events. The basic events supported by the event model are: *mgrget, mgrset, eventget, timer, poll, activate and deactivate.* Of these, all events except *activate* and *deactivate* are system events and cannot be generated by the user. When an event occurs, the event id and event specific details are made available to the receiving cell.

An *mgrget* event is generated when a manager issued **Get Request** is received by the proxy. The cell whose OID is part of the request is the recipient of this event. An *mgrset* event is similar to the *mgrget* event except that it is generated when a manager issued **Set Request** is received by the proxy. An *eventget* event is generated when, as part of event processing, a request is generated for executing the script contained in a cell. The *mgrget, mgrset* and *eventget* events are implicitly enabled for all cells in the spreadsheet. A user of the spreadsheet will have to explicitly disable these events, if necessary.

A *timer* event is generated on every clock tick. A user can optionally specify a time value as part of the timer event specification. A non-zero value indicates that the timer event specification is not true on every time tick, but is only true on those ticks that align with the time interval specified. For example, if a timer event specification contains *timer(5)*, it implies that the event condition will be triggered every 5 seconds . This feature is useful to perform the periodic, repetitive tasks that are typical in network management applications. A *poll* event is generated when a poll response is received by the proxy for an OID that is contained in a cell. The poll event contains the new value of the variable.

A cell in a spreadsheet is capable of generating the following events: *value change, event occurred, invalid value* and *error event.* A *value change* event is generated when a cell's value changes. This event is useful in detecting value based event specifications. An *event occurred* event is generated when the event condition specified in the cell has occurred. This event is useful to trigger dependent cells in an event cell dependency hierarchy. An *invalid value* event is generated when one or more values contained in the cell become invalid due to the dynamic nature of the variables contained in the cell. An *error* event is generated when an error is encountered during script processing.

To support the event model, all cells in a spreadsheet are modeled as *event cells* (i.e.,

they support event generation and receipt). *Event cells* are divided into two categories: *executable cells* and *event-based cells*. The control part of a cell is composed of two parts: an implicit or explicit *event-specification part* (or event expression) and an *action part*. The event-specification part acts as a filter that looks for one or more events to have occurred before executing the action part of the cell. An event is specified using the **on** SSL keyword. The action part of the cell is a set of SSL statements (also referred to as a *script*) that can be executed to perform some management function or subset thereof.

The general structure of a cell that is based on an event condition is:

```
on: (event_expression) ';'
action: SSL_Statement_block
```

where *SSL_Statement_block* is a set of valid SSL statements. The event expression could be a basic event or a derived event and is similar to a boolean expression. Basic and derived events can be combined using the boolean operators defined in SSL (i.e., ||, &&, !). When used in conjunction with the **on** keyword, the expression is treated as an event.

Executable cells allow a manager to request the execution of a script and return the value of the cell that results from the execution of the script. This is a synchronous operation and corresponds to the traditional SNMP framework manager-agent interaction except that a down-loaded script is executed on the proxy before a value is returned. In an executable cell, the following basic events are automatically enabled: 1) manager get request and 2) event processing. Thus, a cell containing executable SSL statements could be executed both by manager request and as part of event processing. A variation on this is the *timed-execution cell* that permits a user to set up a periodic execution of a cell based on some time criteria.

An event-based cell is a refinement of the executable cell. The refinement is in the event specification. A user can specify either basic or derived events as triggering criteria for the action part to execute. This allows a user to use previously defined events in the spreadsheet as triggering criteria. The user can thus build a hierarchy of events, all of which are based on the set of basic events supported by the event model.

The event model operation is shown in Figure 4. When a basic event occurs, the *event dispatcher* forwards the basic event to an *event scheduler*. The event scheduler schedules the cell execution based on priority and other criteria that may be specified for event selection and execution. The event scheduler identifies the target cell for the scheduled event and forwards the event to the appropriate cell or cells. This event, when received by the target cell, causes that cell to process its event condition. If the event condition matches, the action part of the event cell is executed. As part of the execution, other cells may be scheduled for execution and new events may be generated. These events are again forwarded to the event scheduler for further processing. Thus, when a basic event occurs, the cells that are dependent upon the basic event or on events generated as part of event processing are scheduled for execution. This event processing propagates until all the cells that are dependent upon the basic event or on events generated as part of the event processing have completed their execution.

Events can be combined using logical operators to form new events. However, it is important to define the semantics of such event expressions. The parsing of event expressions has to be different since events are dynamic and can occur in any order. The processing of events is contingent upon the occurrence of the event. For instance, if an event expression

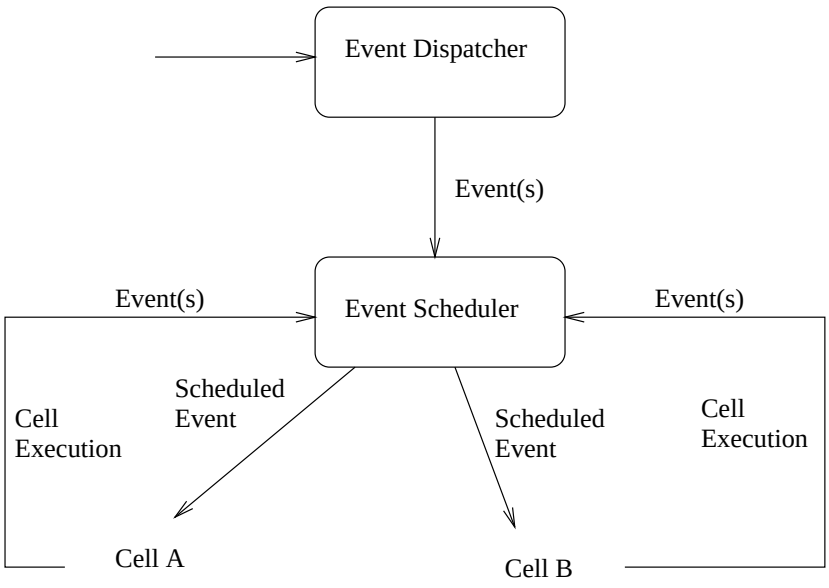

Figure 4 Event Model Support for the Spreadsheet Paradigm

such as *event1* || *event2* is specified, and further let us assume without loss of generality that *event1* occurs at time *t1* and *event2* occurs at time *t2*, and $t1 < t2$ (i.e., *t1* chronologically precedes *t2*), then the derived event *event1* || *event2* is signaled at time *t1* when *event1* occurs. However, the event expression *(event1 && event2)* is true only at the time instant when both *event1* and *event2* have occurred.

6 AN EXAMPLE

The following simple example helps to illustrate the essential features of the SSL vis-a-vis the event model, more specifically timed execution, event hierarchy and event processing. In the example, the number of connections established to certain TCP ports of a host computer is calculated. When a threshold value is crossed, an alarm is triggered and the manager is notified.

```
Cell [1,1]: Periodically download tcpConnTable at host Stimpy.
    label: tcpConnTableStimpy;
    on: timer(5 min); // Download this table every 5 min.
    action: $$ = table tcpConnTable@stimpy.eecis.udel.edu;

Cell [10,1]: TCP port to which the number of connections is calculated.
    label: localPort;

Cell [10,2]: Using the downloaded tcpConnTable of host Stimpy, calculate
     connections to the local port specified in cell localPort.
    label: countPort;
    action:
```

```
    {
     $$ = 0;  // Init count
     foreach $1 in tcpConnTableStimpy do
         if ($1.tcpConnState == 5) // Established(5)
         && ($1.tcpConnLocalPort == localPort) then
     $$ = $$ + 1; // ++ count
         endif;
     done;
    }

Cell [2,1]: Periodically calculate connections established to FTP port.
    label: countFTPConn;
    on: timer(10 min); // Execute this script every 10 min.
    action:
     localPort = 21; // FTP port is 21
     exec countPort; // Execute script of cell countPort
     $$ = countPort; // Get result of execution

Cell [3,1]: Periodically calculate connections established to HTTP port.
    label: countHTTPConn;
    on: timer(15 min); // Execute this script every 15 min.
    action:
     localPort = 80; // HTTP port is 80
     exec countPort; // Execute script of cell countPort
     $$ = countPort; // Get result of execution

Cell [3,2]: Alarm cell - when there are more than 50 connections established
     to HTTP port of host Stimpy, inform the management station.
    label: countHTTPConnAlarm;
    on: countHTTPConn > 50; // When number of connections > 50 do action
    action: inform mgr1@sol.cis.udel.edu HTTPConnAlarm;
// HTTPConnAlarm is the event ID assigned for this event

Cell [3,3]: Enable the above alarm.
    label: activateHTTPAlarm;
    action: activate countHTTPConnAlarm;

Cell [3,4]: Disable the above alarm.
    label: deactivateHTTPAlarm;
    action: deactivate countHTTPConnAlarm;
```

The timed-execution cell tcpConnTableStimpy (cell [1,1]) periodically downloads the tcpConnTable of host Stimpy. Based on this, two other timed-execution cells (cell [2,1] and [3,1]) periodically calculate the number of FTP and HTTP connections using the executable countPort cell (cell [10, 2]). Cell localPort (cell [10,1]) is used by countPort to know which port to calculate on. If the countHTTPConnAlarm cell (cell [3,2]) is in the activated state, whenever the number of HTTP connections is greater than 50, the manager is informed.

Cells [3,3] and [3,4] illustrate the dynamic configuration aspects of the spreadsheet paradigm. A manager can enable or disable countHTTPAlarm and consequently allow or disallow the corresponding event to be reported.

7 CONCLUSIONS

In this paper, a spreadsheet paradigm for network management that uses a proxy architecture along with the motivation for choosing such a paradigm is presented. The components of the spreadsheet paradigm (including a proxy MIB (ssmib), spreadsheet language (SSL) and an event model) are described. The design of a proxy MIB to support the spreadsheet paradigm is covered. The important features of the language (SSL) are outlined followed by the architectural aspects of the event model. Examples that illustrate the essential features of the language have been provided.

A prototype implementation of the spreadsheet paradigm based in a proxy agent is currently in progress. The implementation includes a graphical user interface that can be used by a manager to construct, load and execute scripts in the proxy. Real world applications that can be implemented using the spreadsheet language are also being explored. Future research efforts will include a detailed investigation of the performance and security aspects of the paradigm. Currently, the event model does not support temporal operators and logical ordering of events. Such temporal aspects will improve the power of the event model. Although the history mechanism can be built using the polling and cell related features provided, the actual semantics of such a history mechanism need to be finalized.

In developing this paradigm, the following major goals have been realized: 1) the paradigm facilitates user customization of management information irrespective of the underlying information structure; 2) a distributed network management environment is created that allows the manager to successfully delegate routine management tasks.

The paradigm presented in this paper does not modify the existing management framework, but augments it. The authors hope that the spreadsheet paradigm will deliver a powerful, flexible platform for script based network management in an SNMP framework.

REFERENCES

[AY95] K. Arai and Y. Yemini. MIB view language (MVL) for SNMP. In A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, *Integrated Network Management IV*, pages 454–465. Chapman and Hall, London, 1995.

[CFSD90] J. D. Case, M. S. Fedor, M. L. Schoffstall, and C. Davin. *Simple Network Management Protocol (RFC 1157)*, May 1990.

[CJS93] T. Clifford, G. Jajodia, and S. Snodgrass. *Temporal Databases: Theory, Design and Implementation.* Benjamin Cummings, Redwood City, CA, 1993.

[CMRW96] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2) (RFC 1905)*, January 1996.

[Gol95] Germán Goldszmidt. *Distributed Management by Delegation.* PhD thesis, Columbia University, New York, NY, 1995.

[GY95] Germán Goldszmidt and Yechiam Yemini. Distributed Management by Delegation. In *The 15th International Conference on Distributed Computing Systems*, Vancouver, Canada, June 1995.

[Has95] M. Hasan. An Active Temporal Model for Network Management Databases. In A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, *Integrated Network Management IV*, pages 524–535. Chapman and Hall, London, 1995.

[Hol89] D. Holden. Predictive Languages for Management. In *Integrated Network Management I*, pages 585–596. North Holland, Amsterdam, May 1989.

[KSS96] P. Kalyanasundaram, A.S. Sethi, and C. Sherwin. Design of A Spreadsheet Paradigm for Network Management. In *Proceedings of the 7th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.

[RM90] M. T. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP based internets (RFC 1155)*, May 1990.

[RM91] M. T. Rose and K. McCloghrie. *Management Information Base for Network Management of TCP/IP based internets: MIB-II (RFC 1213)*, March 1991.

[SK94] A. S. Sethi and P. Kalyanasundaram. A Spreadsheet Paradigm for Network Management. In *Proceedings of the 5th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Toulouse, France, October 1994.

[Wal95] S. Waldbusser. *Remote Network Monitoring Management Information Base (RFC 1757)*, February 1995.

[YGY91] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 95–107. North Holland, Amsterdam, 1991.

Pramod Kalyanasundaram received his B.E. degree from the University of Madras, India in 1985 and his M.S. degree from the University of Delaware in 1992. He is currently working towards his Ph.D degree in Computer Science at the University of Delaware. His research interests include network management, operating systems, distributed systems and computer network protocols. He is also a member of IEEE.

Adarshpal S. Sethi received his MS and PhD degrees from the Indian Institute of Technology, Kanpur, India. He is currently Associate Professor in the Department of Computer & Information Sciences at the University of Delaware where he has been since 1983. In the past he has been a Visiting Scientist at IBM Research Laboratories, Zurich, Switzerland. His research interests are in network management protocols and architectures, congestion control and resource management, and protocol testing. Dr. Sethi was Co-Chair of the Program Committee for ISINM '95. He is a member of ACM and the IEEE Computer and Communication Societies.

Chris Sherwin is currently working toward his B.S. degree in Computer Science at the University of Delaware. His research interests include networks, operating systems and distributed systems.

Dong Zhu received his B.S. in Computer Science from Chongqing University, P.R.China in 1988, and M.S. in Computer Science from University of Delaware in 1996. He is currently working towards his PhD degree in Computer Science at the University of Delaware. His research interests include network management, network protocols, and formal specification of protocols. He is a member of ACM.