In Proc. DSOM'96 (Sixth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management), L'Aquila, Italy (Oct. 1996).

Design of a Spreadsheet Paradigm for Network Management [1]

Pramod Kalyanasundaram, Adarshpal S. Sethi, Christopher M. Sherwin

(email: kalyanas@cis.udel.edu, sethi@cis.udel.edu, sherwin@cis.udel.edu)

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

Keywords: Network Management, SNMP, Spreadsheet, Proxy Agent, Intermediate Manager, Internet Management

1 INTRODUCTION

A *Proxy* is a special agent that acts as a front-end for one or more nodes that need to be managed and provides pass-through capabilities. It provides a common mode of access and services to manage nodes that may belong to either the same or a different management model. The proxy does not support any additional functionality vis-a-vis network management. Although proxies are nodes which can provide an interface between two different management environments they are used primarily to bridge administrative incompatibilities. Proxies thus provide a platform for interworking different security models, administrative models, and domain specific policies.

A proxy can also play the role of an "intelligent" intermediate manager. The concept of an intermediate manager exists in network management literature [CMRW93]. However, an intermediate manager – as it exists – only allows delegation of routine management tasks without providing any enhancement to the existing services supported by the management framework. A proxy can play the role of an intermediate manager while implementing additional services which effectively improve the underlying management framework.

A *spreadsheet paradigm* has been introduced in [SK94]. This paradigm incorporates value added functions at the intermediate manager and allows user configurability of management information and control. The main objectives of this paradigm include:

- to introduce a powerful intermediate manager.
- to provide an environment that supports user configurability of management information.
- to support basic primitives, events and operations that allow a user to build fairly complex network management tasks.

[1]Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002.

- to allow dynamic relationships between arbitrary objects.

The power of this paradigm stems from the fact that a user decides to define information and control that are relevant to the user's requirements. The paradigm also allows a user to store and correlate information from several distributed MIBs. This paper presents the design of a spreadsheet entity that implements the spreadsheet paradigm in the SNMP framework.

This paper is organized as follows: Section 2 describes the spreadsheet entity that implements the spreadsheet abstraction; Section 3 presents the design of a spreadsheet language that can be used by a user to set up control in the spreadsheets; Section 4 covers the event model support for the spreadsheet entity and Section 5 summarizes future directions and conclusions.

2 SPREADSHEET ENTITY

The spreadsheet entity that implements the spreadsheet abstraction resides on a proxy and is shown in Figure 1. The essential components of the spreadsheet entity are:

- spreadsheet MIB.
- polling subsystem.
- proxy function module.
- spreadsheet language and interpreter.
- event model.

The spreadsheet MIB (SSMIB) supports the spreadsheet abstraction. A spreadsheet is made up of rows and columns of cells. Each cell has a control and data part. The SSMIB captures the essence of this abstraction and thus has two tables: control table (*controlTable* and data table *dataTable*. The control table can be set up to contain the control portion of a cell and the data table will hold the data values corresponding to the cell. The control and data tables are linked by a *cell descriptor* which is essentially the common index variables between the control and data tables. Figure 2 depicts the essential naming aspects of the SSMIB.

A cell is composed of a row in the *controlTable* and one or more rows in the *dataTable*. Since a cell can contain multiple values, several data table rows may correspond to the same cell. The additional index variable *dataValueIndex* provides unique identification of each value in the data portion of the cell. 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 same common index variable values. This establishes the link between the control and data part of a cell and the cell abstraction is complete.

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. If the manager wants

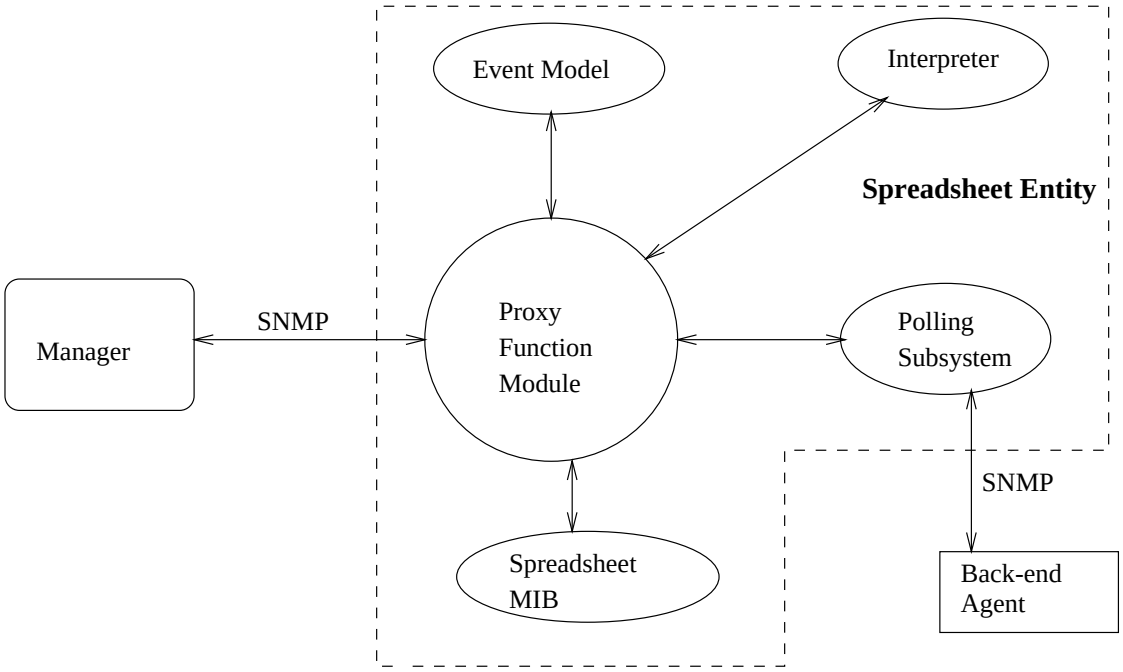

Figure 1: Spreadsheet Entity

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 cell descriptor as a prefix. Figure 3 shows a cell containing four different managed objects in a cell. Such a cell would have one row in the control table and four rows in the data table to represent the four values corresponding to the four managed objects. When the manager retrieves the data values in this cell, the manager gets the four values in order. If the four managed objects are from four different MIBs, then a manager will be able to retrieve the four objects as though they were lexicographically ordered in the same MIB. Thus the user can dynamically restructure the management information and view the information in an order that is independent of the underlying MIBs.

The proxy function module performs the following functions: a) SNMPv2 operations b) MIB access 3) interfaces to the event handling subsystem, polling subsystem and the Spreadsheet Language (SSL) interpreter. When a manager sets up control information in a cell, the proxy function module validates the information and creates an entry in the control table. The control information is passed to the SSL interpreter which parses the script [2]. The parsed script contains information about the number of objects referenced by the script in the cell. The proxy function module uses this information to set up the appropriate number of data rows in the data table. The proxy function module interfaces with the polling subsystem to set up polling entries. This allows the proxy to maintain

[2]The control portion of the cell is also referred to as a script.

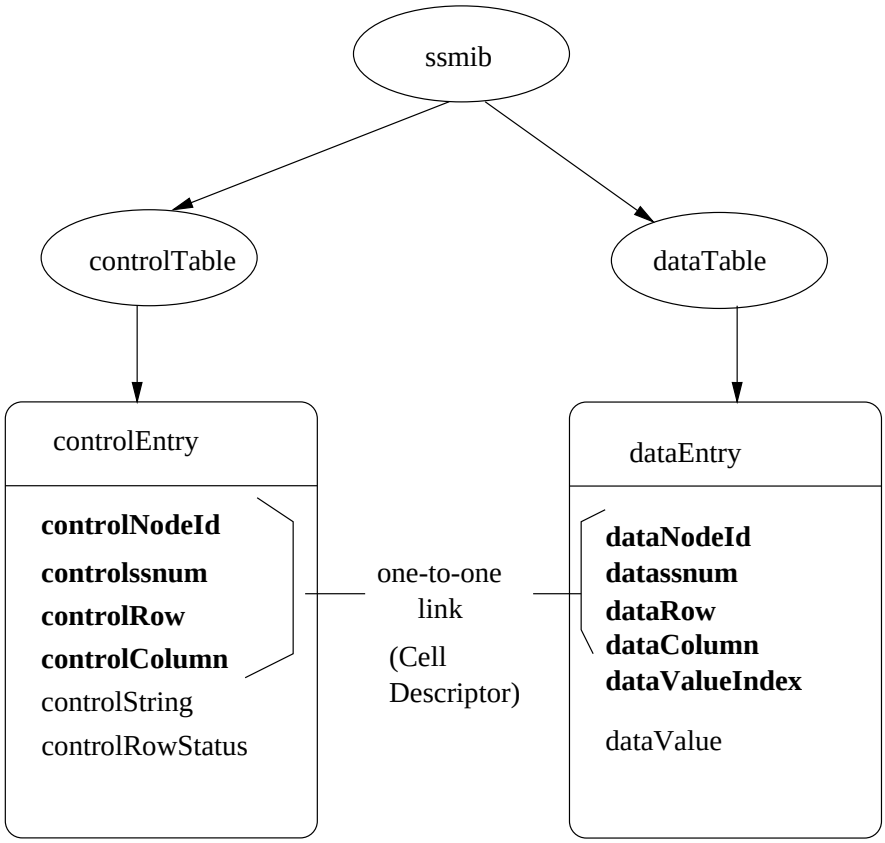

Figure 2: Spreadsheet MIB

a current set of values for the objects referenced by the cell. Once the polling entries are set up, the polling subsystem ensures that periodic polls are initiated to keep the polling entries (and hence the data values in the cell) updated. Once the control and data parts are set up, the manager can query the control or data portion of the cell using the standard SNMPv2 operations. The polling subsystem provides features to group and operate on entries associated with a cell instead of on individual entries. This allows the proxy function module to set up, delete and retrieve all the polling entries associated with a cell.

The spreadsheet supports two modes of operation: synchronous or request/response mode and asynchronous or event mode. 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 and actions associated with such events. These events are constantly monitored by the proxy. On occurrence of any of the events being watched, 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. In order to support asynchronous event processing, the spreadsheet entity uses an underlying event model that permits event and temporal criteria specification.

In order to support the two modes of operation, the cells in the spreadsheet are defined as either *event cells* or *executable cells*. Event cells are those cells that can only have a boolean value. These represent a signaled or non-signaled state of an event. Event

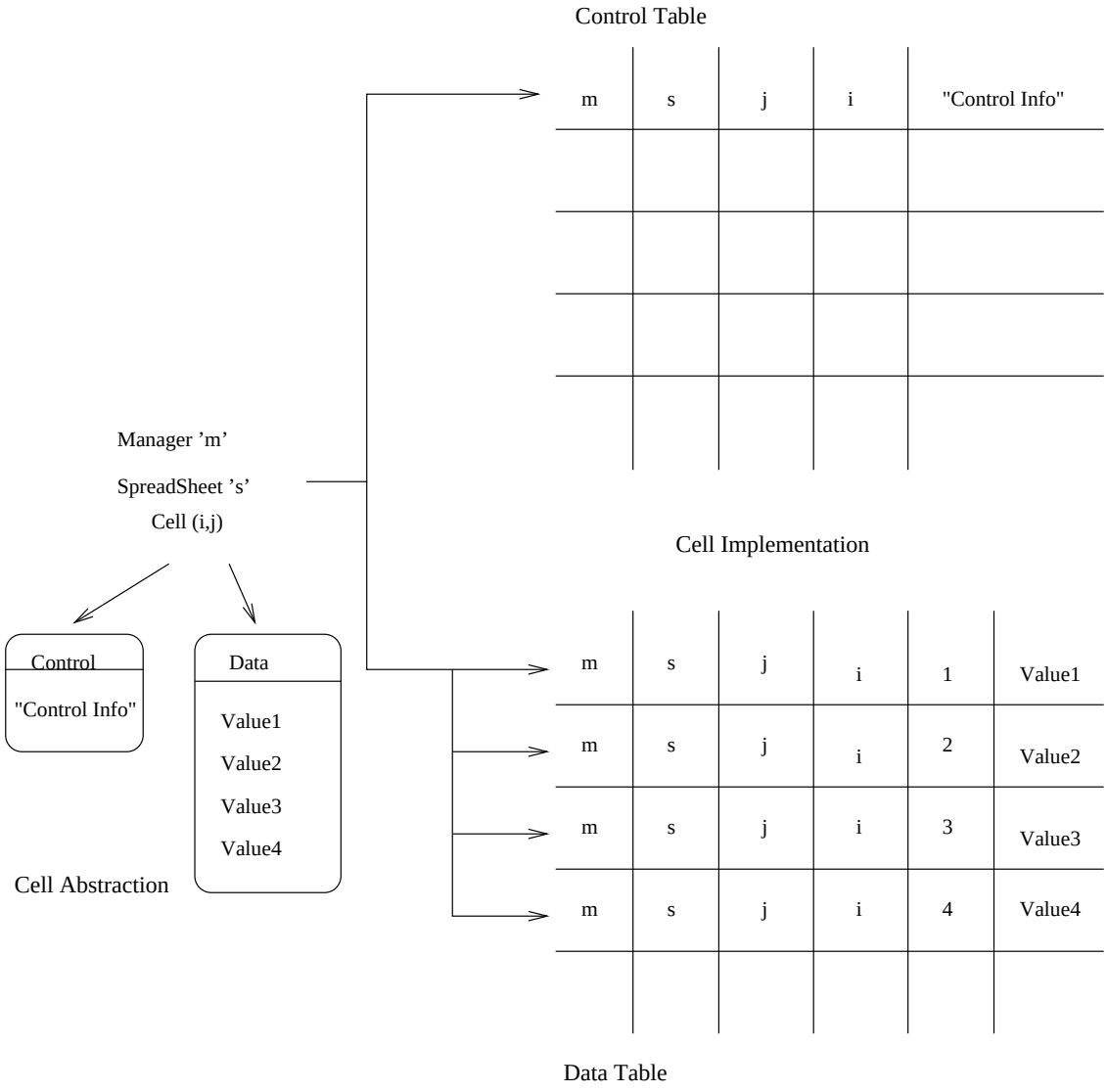

Figure 3: Realization of the Cell Abstraction

cells cannot be executed on request from a manager and are thus part of asynchronous processing only. Such a cell will be triggered on the occurrence of an event in the system. However, a manager can retrieve the value of an event cell which will reflect the current value of the cell.

Executable cells are needed to support the synchronous mode of operation. Executable cells, by definition, contain only statements from the SSL. This allows anyone to invoke an executable cell to run the script that is part of the cell. Executable cells can contain one or more values. This type of cell can be set up by a manager and executed on request or executed by reference during event processing.

3 SPREADSHEET LANGUAGE

In order to provide the user with a flexible, expressive framework for specifying control information, a language is an obvious choice. The spreadsheet language (SSL) provides features that allow a user to set up flexible dynamic control in the spreadsheets. The primary objective in designing the SSL is to provide the user with enough functionality to perform management tasks and not to design an all encompassing programming language.

The main features supported by the SSL include: expression evaluation, control flow, cell addressing, OID specification, event model support, temporal operators and table access. Of these, the event and temporal operators are covered in Section 4.

The SSL supports arithmetic (+,-,/,*), logical (&&, || and !) and relational (>, <, ! =) operators. These operators allow a user to set up dynamic relationships between arbitrary objects. For instance, the arithmetic operator + allows a user to sum a set of counters across agents. The relational operators allow a user to set up event criteria. For instance, if there is a retransmission counter object R, then a link fault could be specified as the condition where retries exceed 5. This situation can be represented by $(R > 5)$. The logical operators allow a user to set up complex conditions which are a combination of several simple conditions formed using arithmetic and/or relational operators.

One of the important features supported by the spreadsheet paradigm is the ability to view management information across nodes. The SSL OID specification which includes host information. This allows a user to easily specify the managed object and the location of the managed object. Thus the same managed object on different nodes would have different specifications. For instance, the managed object *tcpActiveOpens* on the nodes *stimpy* and *dewey* would be represented as *tcpActiveOpens@stimpy* and *tcpActiveOpens@dewey*. Such an OID specification permits a user to access management information anywhere in the management domain.

The SSL provides syntactic representation for cells since cells form the primary unit of computation in the spreadsheet paradigm. This allows a user to retain the spreadsheet abstraction while performing network management tasks. A user can specify cells using a [spreadsheet:row:col] specification. 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. A user can also specify a range of cells.

A cell can contain a single value or a set of values. The SSL provides support for multiple values in a cell by defining set up and access mechanisms for such values. A user can store multiple values by specifying a list of objects and assigning them to a cell. The SSL allows a user to access the nth value in a cell [s:r:c], using [s:r:c].n. Being able to store multiple values in a cell, allows a user to set up a summary of a set of counters and access them easily. Thus a user could set up the *tcpActiveOpens* counter residing on different nodes into a single cell. Once these counters have been set up, a user can retrieve the summary by getting the values in the cell. The SSL provides certain cell variables that can be used for temporary storage by a script that is stored in a cell. These variables are denoted by *$n* where n represents the nth local variable. There is also a special variable $$ which represents the value(s) of a cell. By assigning values to the special $$ variable, a user can set the value of a cell.

The SSL also provides some standard constructs for control flow. These constructs

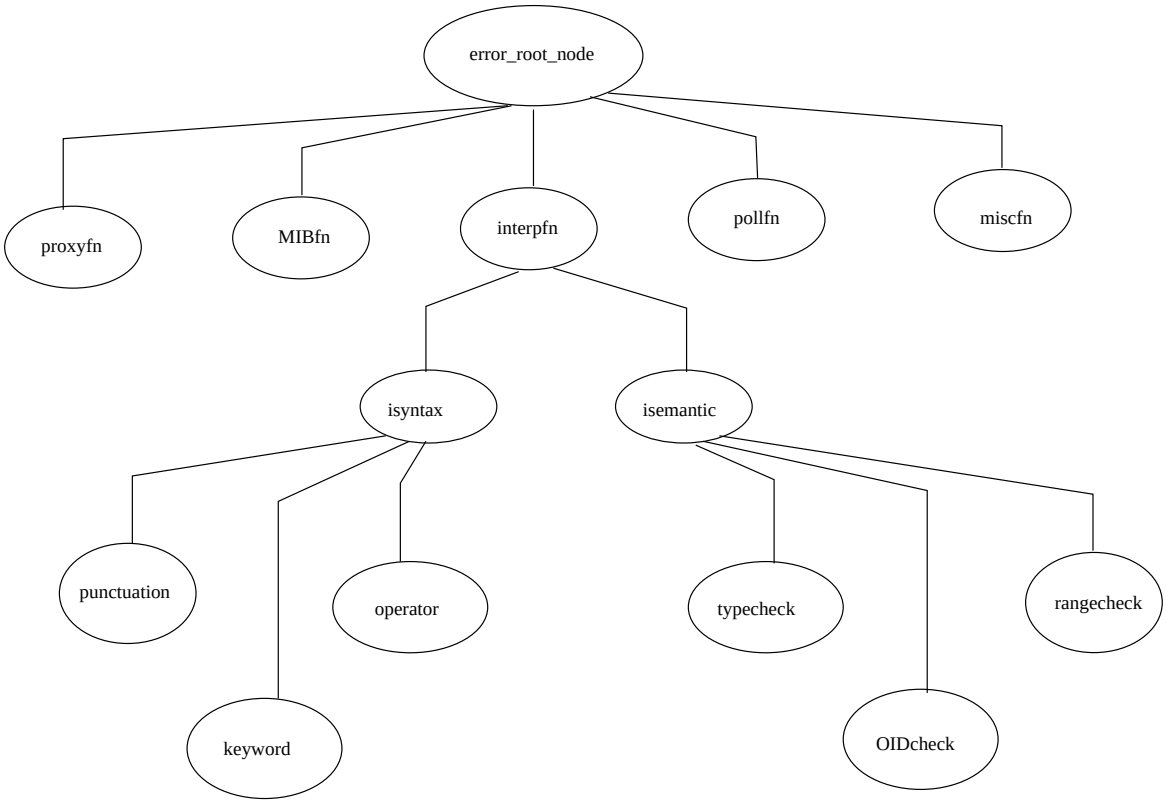

Figure 4: Run time error generation scheme

include condition (*if...else...endif*) iteration (*while*) etc. In addition to control flow, the SSL provides a table loading feature. This feature allows a table to be loaded into a cell. The syntax for this operation is: $table < table - name > @hostindex =< index - val >$. The default value for the index is to get all the values in the table. If a specific value is specified, then only the row of a table that matches the index value specified is retrieved.

Since the SSL is an interpreted language, the language run-time environment has to perform type checking to ensure that compatible objects are involved in expression evaluation. For instance, arithmetic expressions and events cannot be mixed in an expression. The language run-time must therefore retain type information about the current value of a cell.

All error generation is done at run time. The only errors detected and generated off line is syntax related. Since the error codes generated at run time will have to be communicated to the manager, special OIDs are used to convey errors. These OIDs are part of the OID tree shown in Figure 4. This mechanism is primarily to keep the error returning method compact and coded. The error could be returned as an error description string. However, with the OID scheme, the manager interprets the error OIDs and displays a suitable user error message. The tree shown in Figure 4 is fixed with additional entries being added to the tree as the errors get defined. These errors are incrementally numbered under the appropriate node. For instance a specific type of semantic type check error will be listed under isemantic/typecheck/xxx where xxx is the unique error number (relative to typecheck) that is assigned for the error.

4 EVENT MODEL SUPPORT

An event model provides a framework for the specification of events. The model itself supports a limited set of built in events. All other events are user definable. These user definable events are derived events that are created as a combination of the basic events and/or other user defined events. The spreadsheet entity consists of an event model that facilitates the user specification of events. The SSL provides the necessary operators and constructs to define new events.

The need for an event model in network management environments is well established. The polling model is not powerful enough to capture transient faults and behavior. Several research efforts [Hol89, Has95] address event specification models that involve temporal operators and the issues associated with them. This research adapts such work to suit the spreadsheet paradigm and fit into a standard network management framework (i.e., SNMP framework).

The basic events types supported by the spreadsheet event model are the following:

Poll: This event occurs at the proxy when a response to a polling request arrives. This event is set up by a user using the *poll* keyword and specifying the attributes. Some of the attributes include OID of the object being polled, name of the host where the managed object resides, frequency of the poll, start time, and end time. In case of history type polling and additional parameter (i.e., number of samples to be maintained) is also provided. The default value is one sample (i.e., no history). The syntax for the poll specification is: $poll\ OID@host < start-time(default = now) >< end-time(default = forever) >< number-of-samples(default = 1) >$.

Time based events: In order to support a temporal model and clock based ordering of events it is mandatory to be able to keep track of time units. The event model identifies basic time based events that keep track of seconds, minutes and hours. This allows the user to specify intervals in terms of time units. The second event is created once every second, the minute event every minute and the hour event every hour.

In addition, the event model allows a user to activate and deactivate an event. Ability to activate or deactivate an event facilitates the user to set up flexible control that can change dynamically.

The event model orders all events based on the proxy clock. The time stamped on the responses received from the agents may not be accurate depending upon how far apart their clocks have drifted. Depending upon the clocks of the agents to order events may lead to the events being ordered inconsistently.

The event model supports a basic temporal model. A basic definition related to the temporal aspects is an *interval*. An interval can be defined as: a) a time pair (starting time, ending time) b) a starting time and a time period. In essence, definition (b) covers (a), but (a) is provided more for user convenience and will eventually map to definition (b). The starting time can be specified as an absolute time or as the time of occurence of an event. Thus if an event name is specified in the definition of an interval, the starting time is mapped to the time of occurrence of the event.

The temporal operators defined as part of temporal specification include:

Precedence: This operator ($->$) specifies the precedence relation between events. $A->$ B indicates that event A occurred before event B.

Immediately Before: This operator $A => B$ specifies that event A occurred immediately before event B.

Within: This construct allows the specification of an interval based restriction for an event. Thus A *within* I is true iff event A occurs within the specified interval I. This operator can restrict more than one event if a list of events are specified.

Persist: This construct allows the specification of a persistent event. An event B persists over interval I (denoted by *persist* A I) iff event A is in the signaled state for every time tick in I. Thus the persistent event B, defined using A becomes signalled if A is signalled for every time tick in interval I.

The event model caters to the asynchronous event processing mode supported by the spreadsheet paradigm. When a basic event (e.g. poll event) occurs in the system, this may lead to some event cell changing state from a non-signaled state to a signaled state or vice versa. When the event cell changes state, the execution of the cells based on this event cell is triggered by the proxy which may lead to other cells changing state and/or values. This causes a cascade effect continues until there are no more cells that are dependent upon either the initial event or any of the events triggered by the initial event. The proxy maintains an event dependency graph for the cells and triggers the execution of the dependent cells. If there are no cells that are dependent on the initial event, the initial event is just logged as part of the event ordering mechanism and the event model returns to its idle state waiting for an event.

Figure 5 shows an example scenario. The cell dependencies and execution on event occurrence are shown. The dependencies should be interpreted as B depends on A, E depends on B and D etc. When an event occurs that changes the state of the event cell A, it causes cells B and C to execute. If B changes state and E is dependent on such a state change, cell E executes. This event propagation ends in event cell E, since there are no cells that are dependent on E. The cell C1 is an executable cell that is called as part of the execution of cell C. The execution of C1 completes as part of C's execution and the result of the execution of C1 is returned to C for use.

The event model also provides operators for combining events to form event dependencies. These operators include *eventOR, eventAND* and *eventNOT*. These operators allow events to be combined to form the event dependencies. A user specifies an event using an event name, an event condition, a optional temporal restriction and event associated action. The event name identifies the event. The event condition specifies the condition that must be met for the event to be signalled and the event cell to be executed. This condition represents the dependency aspect of the event cell. The temporal restriction specifies any addition time based condition that needs to be satisfied for the event to be signalled. This implies that even if all the dependent events are signaled, an event may not be signaled if the temporal restriction is not met. Action specifies the SSL statements to be executed when an event reaches the signaled state.

The SSL provides a clean separation between event expressions and non event expressions and the spreadsheet paradigm distinguishes between event cells and non-event cells. Such separation will allow a clean language implementation. For instance, the arithmetic

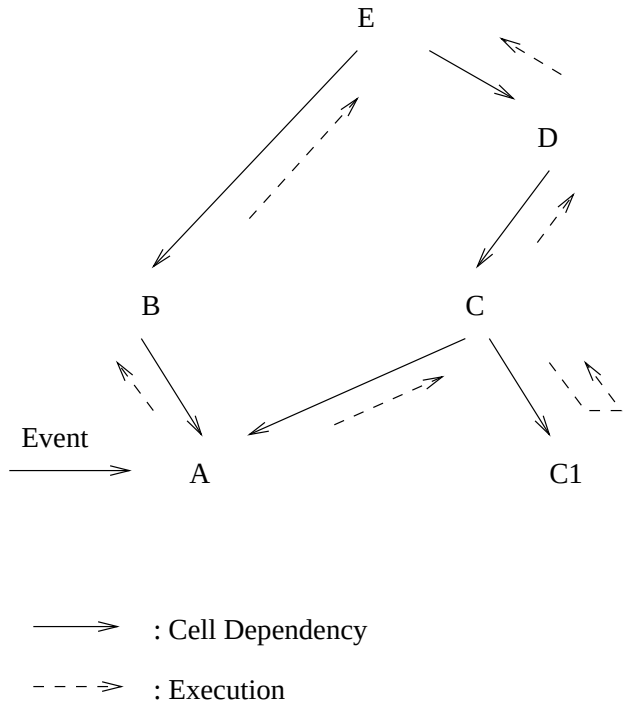

Figure 5: Event Model Example

operators (+,-, etc.) cannot be used with events. The standard control flow constructs like *while) do not work with events. Only event cells can depend on other event cells.*

5 CONCLUSIONS

This paper has presented the design of a spreadsheet entity for network management. The various components and operation of the spreadsheet paradigm are presented. The essential features of a special proxy MIB (SSMIB) are covered. A spreadsheet language and the event model support for asynchronous event processing are described. Future directions include: include the event model supporting constructs into the SSL, implementation of the design presented in this paper, the performance effects of adding the spreadsheet paradigm to the existing SNMP framework, and implementing a few real-world network management tasks using the paradigm to demonstrate its usefulness. The authors believe that this paradigm enhances the power of the existing framework and provides the user with the flexibility of dynamically structuring management information and distributing control. This paradigm thus augments the existing framework by providing value added functions and will prove to be an invaluable tool to network managers.

References

[CMRW93] J. D. Case, K. McCloghrie, M. T. Rose, and S. Waldbusser. *Introduction to Version 2 of the Internet-standard Network Management Framework*, May 1993.

[Has95] M. Hasan. An Active Temporal Model for Network Management Databases. In *Integrated Network Management, IV*, pages 524–535. Chapman and Hall, London, July 1995.

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

[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*, October 1994.