HIERARCHICAL MANAGEMENT OF BATTLEFIELD NETWORKS WITH THE SHAMAN MANAGEMENT SYSTEM

Adarshpal S. Sethi
Dong Zhu
Vasil Hnatyshin

Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716
{*sethi, dzhu, vasil*}*@cis.udel.edu*

ABSTRACT

SHAMAN (Spreadsheet-based Hierarchical Architecture for MANagement) is a novel management framework developed at the University of Delaware as a part of the research in network management sponsored by the ATIRP Consortium. SHAMAN extends the traditional flat SNMP management model to a hierarchical architecture wherein managers can dynamically delegate management tasks to intermediate managers. Tactical battlefield networks require such a hierarchical management architecture to achieve effective real-time management of the large number of mobile nodes that such networks are expected to have. The SHAMAN framework includes a spreadsheet-based intermediate manager with a scripting language and MIB, a polling subsystem, and an event model; a prototype implementation of the system is available. Our research has explored several applications of the SHAMAN system to tactical battlefield networks for the US Army, including a Location Management application and an application to reconfigure dynamically changing topology of Tactical Internets. This paper summarizes the main research results with a description of the SHAMAN system and briefly introduces its applications to the management of tactical battlefield networks.

Keywords: Network Management, Hierarchical Management, SNMP, Tactical Internet, Battlefield Networks, Location Management, Configuration Management.

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.

I. Introduction

One of the significant achievements of the ATIRP Consortium in Technical Factor 2 (Tactical/Strategic Interoperability) has been the design and development of an integrated framework for hierarchical management called SHAMAN (**S**preadsheet-based **H**ierarchical **A**rchitecture for **MAN**agement). This management system developed at the Network Management Laboratory of the University of Delaware incorporates management by delegation concepts into the Internet management framework to facilitate the management of distributed systems and networks [1], [2], [3], [4], [5].

A hierarchical management strategy is an effective means of managing the large and complex internetworks that are in use today [6]. The need for hierarchical management is particularly acute in tactical battlefield networks which are expected to have tens of thousands of nodes, and where scalability is an important concern. Unfortunately, the most popular management framework in use today, the SNMP framework (which includes the family of SNMP, SNMPv2, and SNMPv3 protocols) [7], [8], [9], [10], only supports the flat management model (see Figure 1). The framework provides no means for managers to delegate tasks to intermediate managers or for peer-to-peer communication between intermediate managers during the execution of these tasks.

While the management community in general has tried to design management strategies based on the concept of Management by Delegation (MbD) [11], [12], the SNMP community has not yet been able to take advantage of it because the delegation primitives have not been integrated with the SNMP framework. Our research group in network management at the University of Delaware has designed an integrated framework for hierarchical management called SHAMAN (**S**preadsheet-

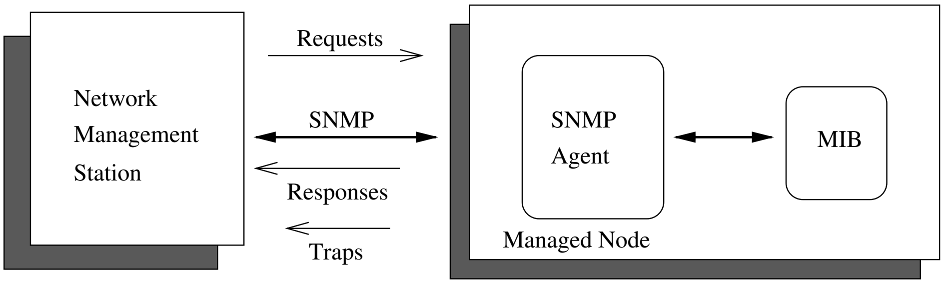

Fig. 1. Internet Management Model.

based **H**ierarchical **A**rchitecture for **MAN**agement) that incorporates management by delegation concepts into the SNMP framework to facilitate the management of large internetworks (Figure 2). This architecture allows a manager to delegate routine management tasks to an intermediate manager and facilitates user configurability of management information and control in a value-added manner. This is achieved by providing a scripting MIB and language specially designed for management tasks in SNMP.

The main objectives of SHAMAN are to introduce a powerful intermediate manager that enhances (but is fully compatible with) the existing SNMP framework, provides value-added functions, supports delegation, allows user configurability of management information, and provides an environment for the rapid development of distributed management applications. SHAMAN captures all the advantages of a hierarchical framework including scalability, increased reliability, possibility of increased fault-tolerant operation, and reduced processing overhead at the top-level manager. By being fully compatible with the Internet standards, it allows existing SNMP managers and agents to take advantage of the hierarchical framework without major changes to the management paradigm or the protocol stack. It further simplifies the task of application development by providing for the specification of dynamic relationships between objects *across MIBs* and permitting flexible, hierarchical event building. Relationship specification allows two or more objects belonging to different MIBs to be related while hierarchical event building allows simple events to be used to build more complex events. These features allow applications to customize MIBs by defining logical views of interest that include relationships between objects across multiple real MIBs and then manipulate and control only those logical views.

A prototype implementation of SHAMAN has been developed at the University of Delaware [13] and a number of applications of the SHAMAN architecture to the management of tactical battlefield networks have been explored [14], [5], [15], [16], [17]. This paper presents an overview of the SHAMAN architecture and its implementation, and describes the battlefield management applications that have been implemented and demonstrated at successive ARL/ATIRP Annual Symposiums over the past five years. Section 2 summarizes existing approaches and other related work in this area. Section 3 describes SHAMAN in detail including some of its main components, namely the Spreadsheet MIB (SS-MIB), the Spreadsheet Scripting Language (SSL), and the event model supported by the spreadsheet paradigm. The prototype implementation of SHAMAN is briefly described in Section 4. Section 5 presents an example application of Location Management in a large ad-hoc tactical battlefield network while Section 6 describes its application to topology reconfiguration of tactical internets. Finally, Section 7 contains the conclusions and outlines future directions.

II. Related Work

Delegation of monitoring and control functions results in decentralizing the network management framework. SHAMAN uses the concept of management by delegation (M_bD), a novel approach to distributed management, introduced in [18]. With M_bD, program fragments – referred to as *agent code* – are transported to special processes called *elastic processes* on one or more servers. These elastic processes provide a framework for dynamic linking, concurrent execution, and control of agent code instances. Thus agent code can be added to or deleted from these elastic processes. M_bD provides a generic conceptual framework for a purely distributed model of management; it does not address the hierarchi-

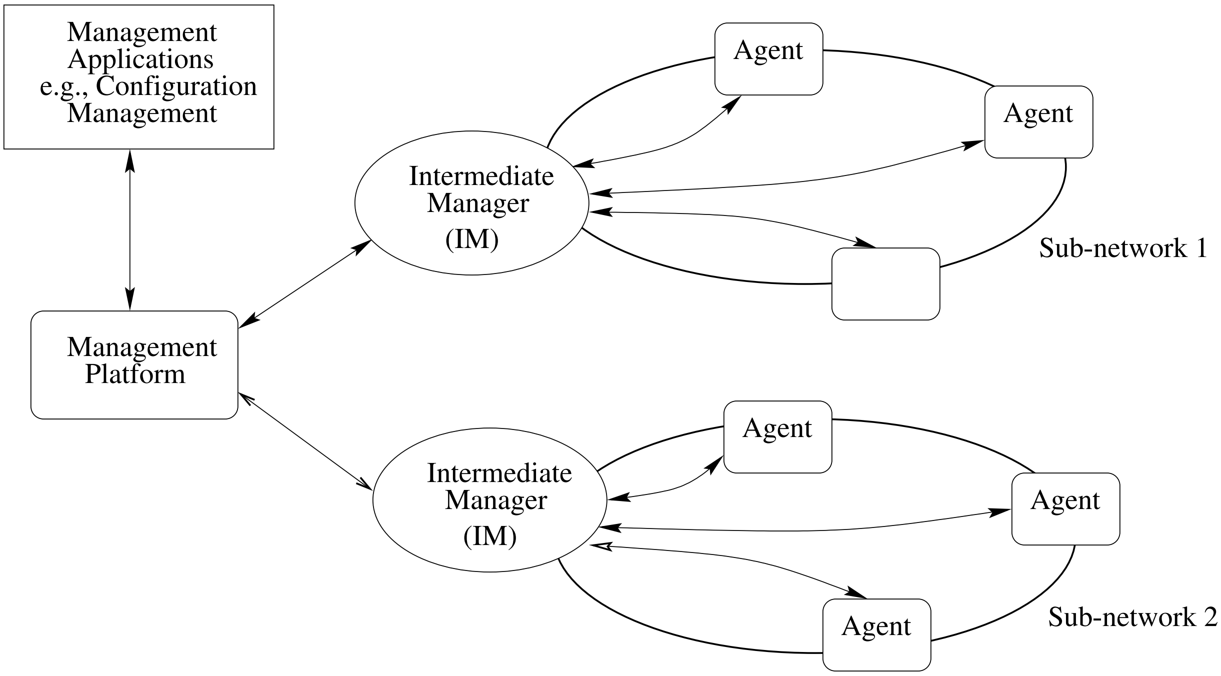

Fig. 2. Hierarchical Management with SHAMAN

cal model, specific language issues for scripting, or event models. The hierarchical model used by SHAMAN was developed concurrently with the evolution of M_bD, but SHAMAN has taken a different form because it provides a specific scripting language, event model, and delegation framework all tailored to work in conjunction with SNMP. Further, M_bD uses the Trivial File Transfer Protocol (TFTP) (contrasted with SNMP used by SHAMAN) to transport the scripts to the agents and is thus not fully integrated within a standard management framework.

Other hierarchical architectures have been proposed to implement scalable frameworks in [19], [20], [21]. A hierarchical model that uses the concept of an intermediate manager to perform rule-based management is discussed in [19]. Rules, which are condition-action pairs, can be loaded into a rule MIB, but the scripting language does not support procedural constructs which limits the expressive power of control specifications. A hierarchical network management system (HNMS) proposed in [21] defines a function based hierarchy but does not address scripting, integration within a standard framework, or event models.

The importance of scripting in management is now being more widely appreciated and there are several research and industry groups [20], [22], [23], including ours, that propose different methods for introducing scripting into management frameworks. The IETF working group for distributed management – DISMAN – [20] has proposed a standard environment for scripting which is based on a set of SNMP MIBs. Each of these MIBs serve a specialized function like script management, event specification, script language specification etc. The DISMAN *event MIB* [24] uses a table to define threshold type events; however, it lacks event dependency specification. The DISMAN approach does not specify the criteria for a scripting language; it limits itself to defining the interfaces needed to support a generalized scripting language for management. This approach uses the help of environments that are external to the management framework to run the scripts.

Minerva [25] is an integrated network management system that supports a custom scripting language, an event driven model for management, and integration with the standard Internet framework. However, this system is resident on a manager and thus has the limitations of a centralized management model. It is also purely event-driven and thus suffers from the limitations of event-driven systems. A distributed architecture for delegating fault diagnosis and testing functions to intelligent agents is discussed in [22]. The intelligent agents support an interpreter that interprets scripts written in a specific scripting language called Scotty. External mechanisms are used for transporting scripts to these agents, and for retrieving the results of script execution.

Our approach differs from DISMAN and the other approaches described above by supporting dynamic structuring of information and control, network-centric view of management information, scripting, and event dependency specifications in a single hierarchical framework using a value-added Intermediate Manager. The power of our approach is the result of identifying the essential components of a scripting environment, and integrating these components into the standard Internet framework, thereby enhancing acceptance. Our approach models the network management processing environment as a collection of related scripts and events that cooperate to achieve a common management function.

In the Internet management framework, any modeling or structuring of management information has to be done using MIBs. Several research efforts [26], [19], [12], including ours, have tried to extend the MIBs to derive power and flexibility in an otherwise flat information model. The pioneering work to extend the capabilities of a MIB using the SNMP structure of management information was RMON [26], [27]. RMON is primarily a MIB specification with associated semantics that are implemented in special agents called *RMON probes* which support the remote monitoring functionality. RMON focuses on the lower level functions in management dealing with link-level traffic statistics, link monitoring, and gathering history information. RMON has achieved some level of delegation, event definition support, and asynchronous notifications. Although our approach uses some concepts derived from RMON, it differs significantly from it due to the addition of a computing paradigm, scripting support, and an event model that supports dependencies. Further, unlike RMON which focuses on lower level functions (e.g., link traffic), our work targets higher level network management functions (e.g., location management).

A MIB View Language (MVL) is proposed in [12] that uses a relational database perspective on data aggregation. MVL extends the standard Internet Structure of Management Information (SMI) to define "MIB Views" which are computations over sets of managed objects. MVL supports the *select* and *join* operations that resemble relational database table operations. MVL integrates with SNMP and permits data aggregation by statically defining the associated computations as part of a MIB. Our approach of using a language to perform the aggregation instead of statically defining such aggregation improves flexibility. A novel approach to adding time as a dimension to extend standard MIBs to store history is explored in [28].

Event specification languages allow specification of events and establishing powerful relationships between events. Several event based languages have been designed for different purposes [29], [30], [31]. All these languages support event specification and combination; they support the specification of a simple or compound event. State Event Specification Language (SESL) [29] is a declarative style language for specifying events with a temporal component that includes operators and time intervals. Another approach used by [32] represents network management functions as $<event, condition, action>$ tuples. Events are specified using an Event Specification Language (ESL); actions are specified using a Data Manipulative Language (DML) like SQL. The focus is on specifying event-based triggers to perform network management database operations. A MIB View Language (MVL) is proposed in [12] that uses a relational database perspective on data aggregation. MVL extends the standard Internet Structure of Management Information (SMI) to define "MIB Views" which are computations over sets of managed objects. MVL supports the *select* and *join* operations that resemble relational database table operations. MVL integrates with SNMP and permits data aggregation by statically defining the associated computations as part of a MIB. Our approach of using a language to perform the aggregation instead of statically defining such aggregation improves flexibility. A novel approach to adding time as a dimension to extend standard MIBs to store history is explored in [28].

The primary focus of the aforementioned languages is to support event specification and event models; they do not address integration within a framework, delegation, development of procedural scripts, or restructuring management information.

All event specification languages support an underlying event model. An event model consists of two parts: event definition and event propagation. There are several industry and academic research efforts [33], [34], [35], including ours [36], that have come up with different event models.

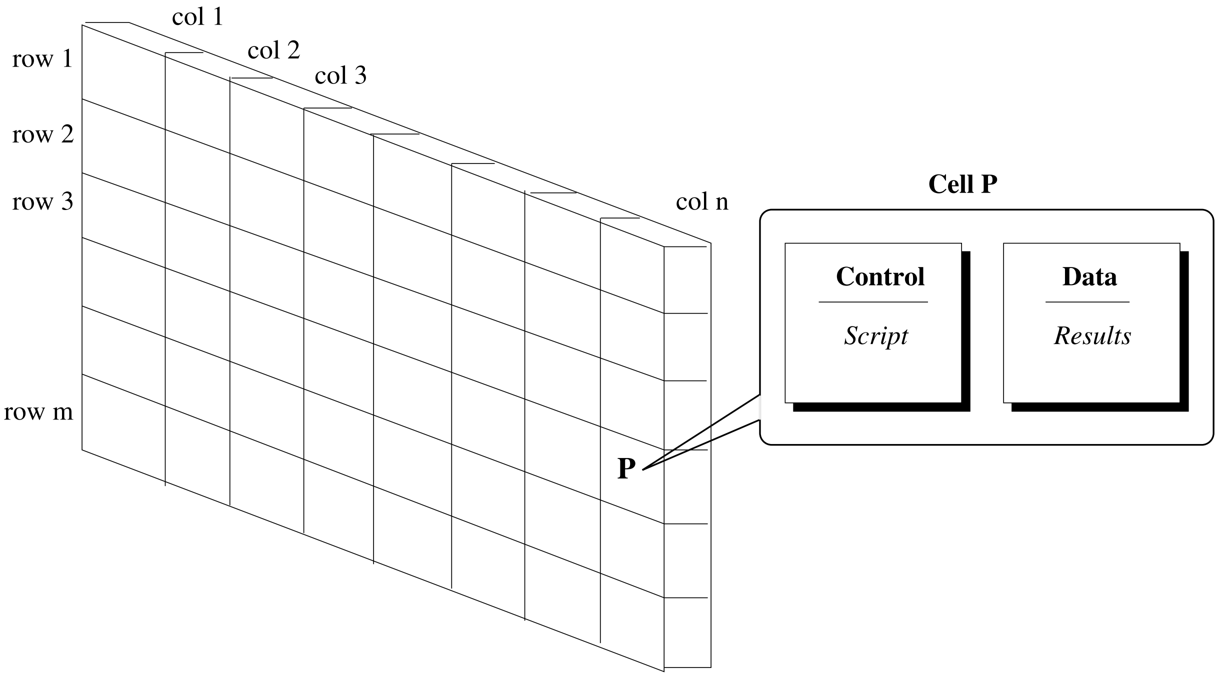

Fig. 3. Spreadsheet Structure in SHAMAN

The OSI set of standards on management include the specification of an event model. The OSI event model defines functions for notification generation and event report management. Events can be generated when management objects are created, deleted or the state of an object changes. When an event occurs it is forwarded to an Event Forwarding Discriminator (EFD); based on a filtering criteria, the EFD decides if an event should be forwarded, and identifies the intended recipient of the event. While primitive data aggregation is possible, the OSI event model does not combine events with scripting. [37] introduces scripting in an OSI environment using pointers to scripts that are stored as part of an OSI object definition.

MODEL [34] is a declarative style language used to specify events and create event models. Using MODEL, an event model can be defined external to the SNMP framework, as a separate layer, with mappings to MIB variables. The focus of this approach is on developing a generalized event model with emphasis on building event correlation systems. A knowledge-based event correlation system is proposed in [35]. This approach primarily targets event correlation, and defines an extensive set of temporal and procedural operators to suit event correlation environments.

III. SHAMAN – A Spreadsheet based Hierarchical Architecture for Management

SHAMAN provides a hierarchical architecture for management by making it possible for managers to delegate management operations to one or more Intermediate Managers (IMs). Each IM can in turn delegate operations to other IMs in a hierarchy of levels, with management agents at the lowest level. SHAMAN is called Spreadsheet-based because it uses the well-known concept of a spreadsheet to structure the operations to be executed in an Intermediate Manager.

A conventional spreadsheet is composed of *cells* arranged in a two-dimensional matrix. Thus a cell is accessed by specifying a row and a column index. Further, a cell contains an optional *formula* (control part) and *data*. A spreadsheet may specify cell value dependencies to implement a *refresh* feature that allows automatic recomputation of formulae that are dependent upon a value that has undergone a change.

SHAMAN views a management task as a collection of cooperating sub-tasks that can be set up in one or more cells in a spreadsheet within an Intermediate Manager. Depending upon the management task, these cells can be related by dependency specifications to form a group of cells that perform the common management function.

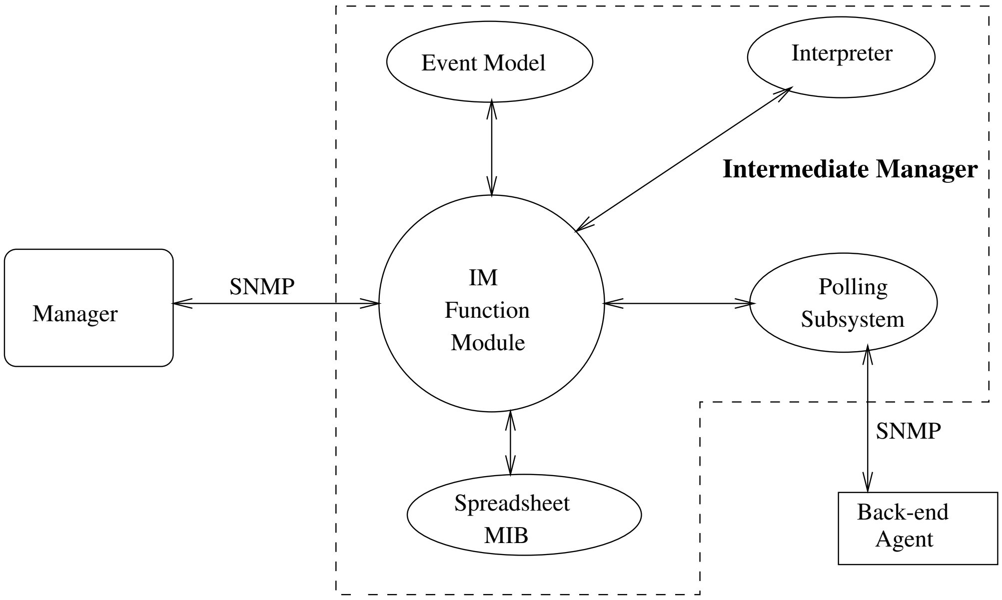

Fig. 4. Structure of an Intermediate Manager (IM)

The power of SHAMAN results from scripting and event hierarchy specifications.

The spreadsheet used by SHAMAN differs in many ways from its well known counterpart. While the SHAMAN spreadsheet retains the two-dimensional matrix structure and access methods for cells, the contents of the control and data portions differ. The control part of a cell contains programs called *scripts* (Figure 3). These scripts are more complex than the *formula* allowed in the well known spreadsheet. Thus, the control part can serve as a repository for event specifications, relationships between management objects, references to other cells, or some computation. The data portion of the cell is also different from the traditional spreadsheet in that it can contain *multiple values*. The cell dependency is enhanced to support both value and event based dependencies. This extended dependency mechanism allows the creation of powerful event hierarchies involving groups of cells to construct a management task. Also, unlike the well known spreadsheet, cells can be created or deleted.

The control part of the cell dictates the rules for collection of information or relationships between objects. The data part of the cell contains the data resulting from executing the script contained in the control part. For instance, the script can 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. A two dimensional structure has been chosen to create a table-like structure which is the only compound data structure supported by the SNMP information model. This is an important feature of SHAMAN because it allows the access mechanism for the cells in a spreadsheet to mimic the retrieval of a conventional SNMP table.

Because SHAMAN is totally compatible with SNMP, standard SNMP protocol operations are used by a manager to create and download scripts to an IM for the purpose of delegating management tasks to it. These scripts are set up in the control portions of the various cells specified by the row and column indices during script creation. A script can contain event specifications that cause the IM to monitor those events. The manager can either poll for or be notified of conditions that are of interest to it. An IM can support multiple spreadsheets. Access control is defined at the spreadsheet level with each manager capable of controlling one or more spreadsheets.

An IM consists of the following main logical components: a communication module, a spreadsheet MIB (SS-MIB), a scripting language (SSL) interpreter, a polling subsystem, and an event model (Figure 4). The communication module handles all communication between the IM on one hand and the managers and the agents on the other. This communication is in the form of SNMP messages that contain PDUs for SNMP requests, responses, and traps.

The spreadsheet MIB (*SSMIB*) implements the spreadsheet using SNMP tables. User operations on cells map to operations on tables that are part of this MIB. When the IM receives an SNMP request from the manager that translates to an operation on a cell, the IM performs the necessary operations on the spreadsheet MIB to implement the cell abstraction. Once the request has been carried out, the IM responds to the manager that requested the cell operation.

When a script is entered into a cell, it is parsed to determine the number of data values in the cell, i.e., the number of managed objects referenced by the script. The data portion of the cell is then created based on the number of data values determined.

Since polling is the primary means of retrieving information in the Internet management framework [38], it is possible to set up managed objects in a cell to be periodically updated via polling from managed objects maintained at an agent. Polling entries are created in the polling subsystem for the managed objects referenced in a script when a script is entered in a cell. Once set up, these entries cause poll requests to be forwarded to the appropriate agents maintaining these managed objects. When the poll responses are received from the agents, the values in the corresponding cells in the spreadsheet are updated. The IM maintains a cache, and therefore a snapshot, of the management information defined by the scripts. As a result, the values seen by the manager in its MIB view are current (within a certain time granularity).

The polling subsystem plays a vital role in SHAMAN since the IM may poll and cache a significant amount of data. An inefficient polling system can unnecessarily load the network [39]. Thus, it is imperative, for a framework like SHAMAN, to support an efficient and optimal polling mechanism. In order to effect polling, the IM maintains polling tables to facilitate polling and automatic update of the cell values. The polling subsystem allows entries to be set up in its polling tables based on a cell id, so it is easy to retrieve the set of values that are part of a given cell.

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 [40]. 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 the minimum frequency of polling that will satisfy all the polling requests for the variable.

SHAMAN supports a scripting language called Spreadsheet Scripting Language (SSL). This interpreted language is described in Section III-B. The language contains features that facilitate the development of procedural scripts as well as event specifications. The language supports the spreadsheet paradigm by providing syntactic constructs for cell access and assignment. A script in this language consists of three parts: 1) an optional initialization part 2) an optional event specification part and 3) an action part. The initialization part allows a one-time configuration for the cell, the event specification is for setting up events in the cell, and the action part is the program that is to be executed when the event condition for the cell is enabled.

The SSL 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 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 IM responds after processing the request. In the asynchronous mode, the script defines events and actions associated with such events. These events are constantly monitored by the IM. On occurrence of any of the events being watched, the IM 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 IM. In order to support asynchronous event processing, the IM uses an underlying event model that permits event and temporal criteria specification. The SSL defines the syntax for event specification and the event model handles event semantics.

A. Spreadsheet MIB (SSMIB)

The Spreadsheet MIB (SSMIB) supports the fundamental spreadsheet and cell operations that are needed within

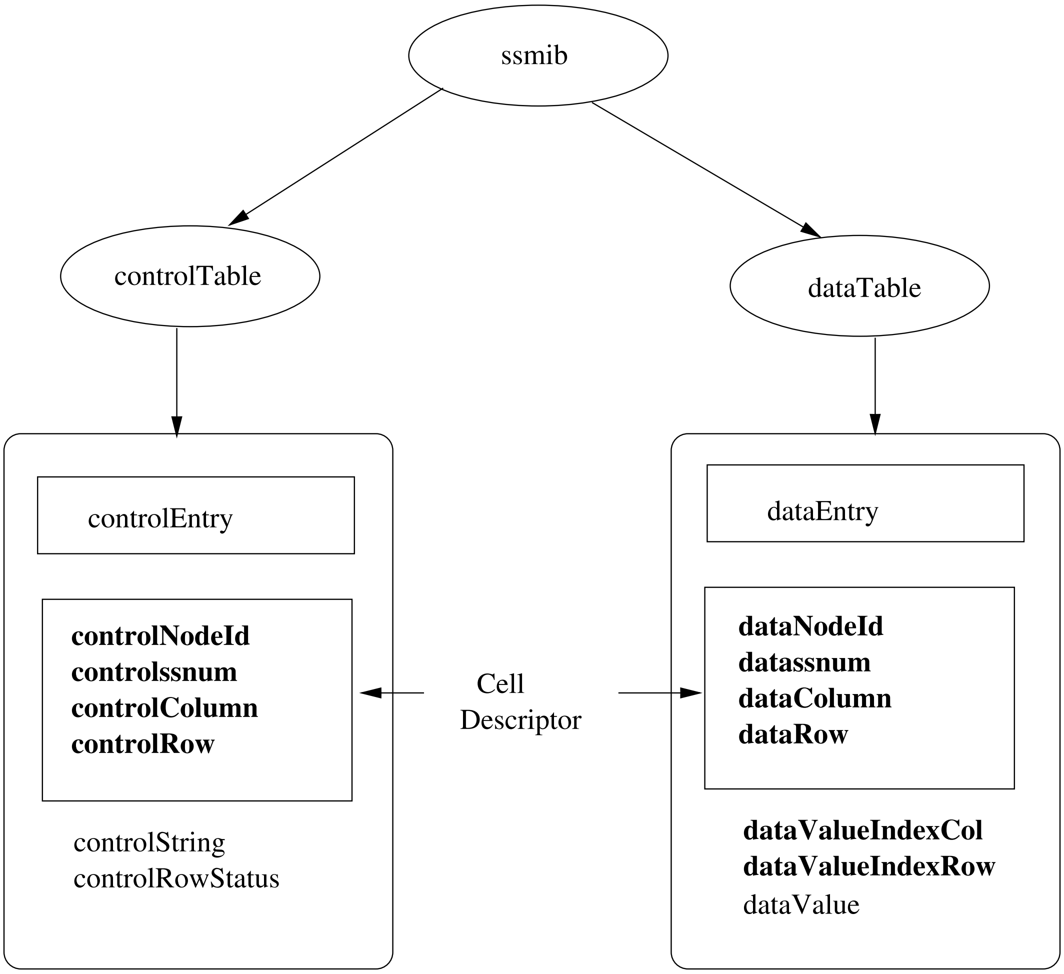

Fig. 5. Spreadsheet MIB

an IM for its operation. A cell in a spreadsheet has a control part and a data part; the control part can contain one script whereas the data part can contain one or more values called *cell-elements*. When multiple values are present, the values may represent a single column or may be two-dimensional (containing both rows and columns).

The SSMIB captures the control and data parts of a cell using two tables: a control table and a data table. Each cell is represented by a row in the *controlTable* (the control part) and one or more rows in the *dataTable* (the data part). The structure of the MIB is shown in Figure 5. The control table (*controlTable*) is made up of a sequence of *controlEntry* elements each of which contains the column variables that constitute a row in the control table. Since the control and data tables are mapped to SNMP tables, the semantics of the SNMP table access apply to these tables. A table row in an SNMP table is uniquely identified by an index value. The index is specified by one or more column variables that form part of the row specification. In the case of the control table, the index is defined by *controlNodeId*, *controlssnum*, *controlColumn*, and *controlRow*. The values of these variables together uniquely identify the control portion of a cell. The variable *controlNodeId* represents the primary internet address of the manager who is the owner of the row, *controlssnum* denotes the spreadsheet number, *controlColumn* the column id of the cell, and *controlRow* the row id of the cell. The *controlEntry* also contains a *controlstring* field that holds the control information of the cell. The *controlRowStatus* field controls the creation of conceptual rows in the *controlTable* and has standard SNMPv2 *RowStatus* semantics [41, pages 288-289]. The status of this field determines the availability of the associated conceptual row in the *controlTable* for access by management applications.

Similarly, the data table (*dataTable*) is made up of a sequence of *dataEntry* elements. The *dataEntry* contains the column variables that form a row in the *dataTable*. The data table index is defined by *dataNodeId*, *datassnum*, *dataColumn*, and *dataRow*. The data table index variables are the same as that of the control table with one difference – there are two additional index variables *dataValueIndexCol* and *dataValueIndexRow*. Since the data portion of a cell can contain multiple cell-elements, these additional index variables help to uniquely identify a particular data value.

The control and data parts of a cell are linked using a set of variables called the *cell descriptor* (Figure 5). The cell descriptor is formed by the column variables that are common to both the control and data tables, i.e., the manager id, the spreadsheet number, the column and row ids. The values of the cell descriptor variables uniquely identify a cell in a spreadsheet.

The relationship of the control and data tables with a cell is shown in Figure 6. The left hand side of the figure shows the abstraction of a cell with a cell descriptor $<m, s, j, i>$, where m is the manager id, s is the spreadsheet number, j is the column id, and i is the row id. The control portion of the cell has been set up with a set of counters (*tcpActiveOpens*) from three different hosts named sol, tweety, and stimpy. The data portion of this cell thus has three cell-elements which reflect the last polled value of these counters. The right hand side of Figure 6 shows the mapping of the cell's control and data portions to the control and data tables in the SSMIB. The control portion maps to a single row in the control table and the data portion to three separate rows, one each for each cell-element. The three rows represent the values of the *tcpActiveOpens* counters from sol, tweety, and stimpy respectively.

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 manager can delete a row in the control table, and based on the control to data table association, the corresponding rows in the data table are removed. To modify the control part of a cell or to retrieve its value, the manager performs an SNMP Set or Get operation respectively on the appropriate row in the control table. In order to retrieve the data portion of a cell, the manager can use any of the SNMP Get, GetNext, or GetBulk operations, even when the cell contains multiple data values.

A key aspect of the SSMIB is demonstrated in the example of Figure 6. Although each of the three variables contained in the cell belongs to MIBs in agents at different hosts, they can be retrieved in an order defined by the control portion of the cell. Management information can be restructured selectively and viewed in an order that is different from that of the underlying agent MIBs. Such reordering of management information allows setting up summaries and different views of the management information. This dynamic configuration of the MIB does not exist in the current SNMP management framework.

B. Spreadsheet Scripting Language (SSL)

A language that targets a network management environment must be able to support features that facilitate the specification of network management tasks; such a language should also be simple yet powerful [29], [37]. This section presents the Spreadsheet Scripting Language (SSL) that forms an integral part of SHAMAN and supports the development of network management scripts.

The main features of SSL can be broadly classified as:

- procedural language related features including operators, variable support, and control flow constructs
- network management specific features including polling specification and management variable access
- paradigm specific features including cell access, retrieval, modification, and multiple value access
- event model related features including event and event dependency specification

A detailed description of the SSL grammar is available in [42], [13].

B.1 Procedural Language Features

Table I lists the operators supported by SSL. These include the standard arithmetic, relational, logical, and assignment operators found in most procedural languages with standard semantics [43]. The <> operator expresses the "not equals" relation.

Based on the number of values they contain, variables in SSL are of two types: single or multi-valued. Multi-valued variables are needed for processing table rows which typically consist of a sequence of column values. Thus an SNMP scalar would be stored in a single-valued

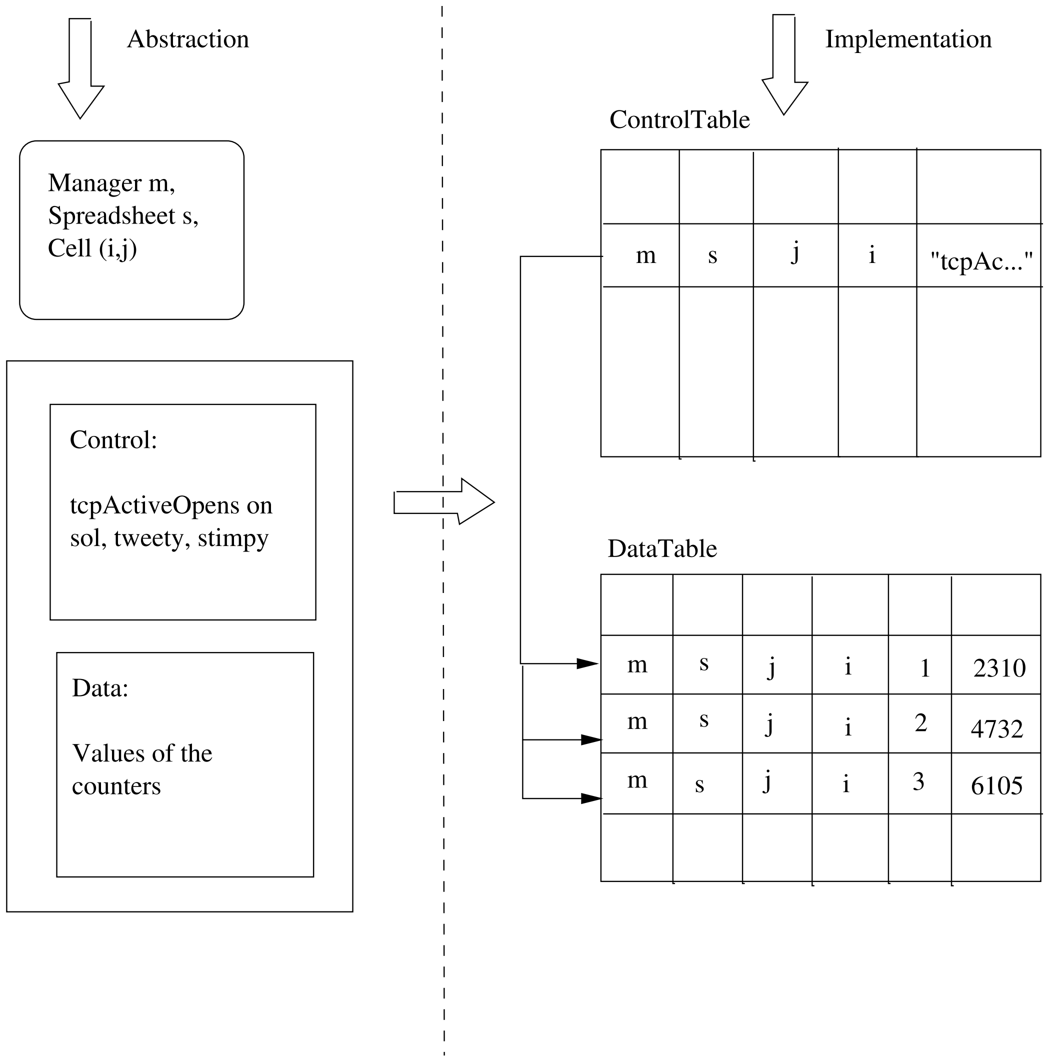

Fig. 6. Cell abstraction and implementation

TABLE I
SSL OPERATOR TABLE

Operators	Type
$+, -, *, /$	arithmetic operators
$>, <, <>, ==$	relational operators
&&, $\|\|$, !	logical operators
=	assignment operator
$\rightarrow$	temporal dependency operator

variable while an SNMP table row is stored as a multi-valued variable.

Based on scope, variables in SSL can be classified as local variables or *cell-valued* variables. Local variables are available in SSL to provide temporary storage. Each cell has a set of such local variables that can be referenced with syntax $\$n$, where n=1,2,...,m, where m is implementation dependent. A local variable – as the name indicates – is visible only within the cell that defines the variable. Thus each cell has its own set of local variables.

A cell-valued variable represents the data portion of a cell. Thus a cell-valued variable can be either single or multi-valued since the data portion of a cell can contain one or more values. Each value in a cell-valued variable is referred to as a *cell-element*. A cell-valued variable is specified using the notation [s:r:c]; the nth value (or nth cell-element) in the cell-valued variable is specified using the notation [s:r:c].n. When a cell has two-dimensional data, the value in row dr and column dc is specified as [s:r:c].dr.dc. For a cell with just one data value, the notation [s:r:c] can be used instead of [s:r:c].1 or [s:r:c].1.1.

While a cell-valued variable can be referred to by other cells using its full name, a convenient shorthand name, $$, can be used by a cell to refer to its own data portion. This notation does not represent a different variable and therefore has the same semantics as [s:r:c], except that its scope is limited to its own cell. For instance, a cell [1:2:3] could modify its first cell-element using $$.1. This modification is then visible to the other cells through [1:2:3].1. For cells with only a single value, the notation $$ may be used.

There is no explicit declaration or definition of variables in SSL. The type information is part of the variable and type-checking is deferred until run-time. This is due to the fact that variables in SSL can be dynamically bound to different types at run-time. Thus static type-checking is not possible.

SSL includes assignment, iteration, conditional, and management specific statements. The left hand side of an assignment statement can only be a single-valued variable. The right hand side can be any expression that results in a single value.

Control flow constructs like **if...else...endif** and **while** are used to set up conditional and iterative scripts in cells. The semantics of these constructs are similar to those found in standard procedural languages. However, iteration has an implicit, user-configurable, maximum loop count that ensures that a single script will not run forever. SSL supports a special iterative construct for processing lists of variables, called the **foreach** statement.

The **foreach** statement views a multiple-valued variable (local or cell-valued variable) as a sequence of values that can be stepped through one at a time to perform an iterative computation. For instance, if there are multiple values in a cell, the **foreach** statement can be used to iterate through all the values and perform some computation. The example script shown in Figure 7 sums all the values contained in the cell [1:1:2].

```
foreach $1 in [1:1:2]
            // for all variables in cell 1:1:2
do
   $$ = $$ + $1; // sum and store value locally
done;
```

Fig. 7. Example usage of **foreach** statement

B.2 Network Management Specific Features

With SSL, a fully qualified managed object in a MIB can be specified 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 name or address of the host on which it resides. For instance, the MIB-II *tcp* group variable *tcpActiveOpens* on host *stimpy@udel.edu* is specified as *tcpActiveOpens@stimpy@udel.edu*. This feature allows access to any managed object in the management domain.

Use of such a name for a MIB object is permitted inside any expression. When the name is processed during the execution of the expression, an SNMP GetRequest is sent to the appropriate agent for getting the value of the MIB variable. When the name is used on the left-hand side of an assignment statement, an SNMP SetRequest is sent to the agent for setting and modifying the value of the variable. Thus, MIB objects can be used interchangeably with other variables within expressions and statements, and appropriate SNMP requests are transparently generated by the IM.

In some situations, it is preferable to poll a MIB object at a specific frequency and cache the value in the IM instead of explicitly fetching it every time it is needed. The **poll** statement is used to specify the polling of a management variable. The polling frequency and the number of samples to be retained can be optionally specified as part of the **poll** statement.

To support bulk processing, the SSL provides a **table** statement to facilitate loading tables from agent MIBs. A table, when loaded, occupies a single cell as a two-dimensional data value. Optionally, one can specify a subset of the columns of a table to be fetched instead of the entire table. The **foreachrow** and **foreachcol** statements allow iteration over a table's rows and columns stored in a cell.

The **foreachrow** statement is similar in semantics to the **foreach** statement, except that it operates upon a sequence of table rows instead of a sequence of single values. It loops the index variable through successive values starting with 1 and going through the integer index of the last table row. This index variable can then be used along with the cell name to access each row in turn within the loop body. The **foreachcol** statement similarly loops through table columns. The **foreachrow** and **foreach-**

col statements can be nested, thereby permitting access to each individual data value in a two-dimensional table. It should be noted that these statements do not require any prior knowledge of the size of the table; the size information is maintained internally by the IM when the table is loaded and is used transparently in looping through the rows and columns.

B.3 Spreadsheet Specific Features

SSL provides facilities to access and manipulate cells in a spreadsheet, some of which have been described earlier, such as the naming of cells and the data values contained in them. Cells can be given symbolic names which can be used in scripts instead of their numeric names. Other features include the special $$ variable for accessing the data portion of a cell, and the support for multiple cell-element reference and access.

It is possible to execute a script in another cell using the **exec** statement. The **exec** statement permits a management task to be divided into smaller scripts and entered into several cells. The scripts in these cells can be successively executed using the **exec** statement. Thus larger scripts can be supported without being affected by the transport packet size restrictions. The **exec** statement blocks the caller until the callee has completed execution.

The data portion of the cell and local variables needed by a script can be initialized using the **init** clause. For instance, if a set of variables need to be polled or if a table is to be loaded into a cell, the polling or table loading statement can be set up as part of the initialization. Once set up, the variables will be polled at the frequency specified. The **poll** and **table** statements are permitted only as part of cell initialization.

Dynamic configuration of scripts in a spreadsheet can be effected using the **activate** or **deactivate** statements. An **activate** statement causes the script in a cell to be enabled; a **deactivate** statement disables the script in a cell. For instance, consider the application where a set of agents are being polled; the polling should stop on encountering a fault on the path to the agent; polling should resume when the fault is rectified. The manager loads all the scripts onto an IM. When a fault is encountered, either the manager or the IM can disable the scripts that poll the agents affected by the faulty link. When the fault is rectified, the manager can activate the scripts that were previously deactivated and polling resumes. This dynamic configuration is more efficient than deleting the loaded scripts and reloading them. Deactivation causes the polling or table loading in the cell to be suspended while activation resumes such activities. In addition, activation causes the **init** portion of the cell to be executed.

B.4 Event Model Related Features

Event expressions are specified using the **on** clause. From a language syntax specification perspective, the following differences contrast event expressions from boolean expressions:

1. the ! (logical NOT) operator is not allowed in event expression specifications.
2. Temporal criteria are relevant only in the context of event expressions; the $\rightarrow$ operator (Table I) is specific to event expressions only and cannot be used in boolean expressions. This operator expresses chronological precedence of events; thus, $A \rightarrow B$ implies event A occurs before event B.
3. the arithmetic operators are not defined for event expressions.
4. the assignment operator is not defined for event expressions.

Event expression semantics are described in Section III-C. However, a few syntactic aspects are discussed here. The syntax [s:r:c].$*$, when used in an event expression, implies a value change dependency upon any of the cell-elements contained in the data portion of cell [s:r:c]. When a relational construct such as [s:r:c].$n > k$ is used in an event expression, the semantics imply the crossing of a threshold; the boolean expression [s:r:c].$n > k$ checks the value of the variable [s:r:c].n and evaluates to true or false depending upon whether the value is greater than k or not.

To summarize, SSL combines standard procedural language constructs with event specification constructs to provide a simple, yet powerful platform for developing scripts to perform network management tasks.

C. Event Model

The event model consists of a set of events and algorithms for event propagation and dependency processing. The SNMP framework is predominantly synchronous in the sense that most communication has the request-response structure. The transfer of management information from the agent to the manager is achieved using polling [44, pages 40–41]. Polled systems have an inherent disadvantage that information changes and events that fall within the polled interval are lost. The primary source of asynchronous processing in the SNMP environment is the transmission of *traps* from an agent to the manager. The SNMP framework favors the use of a technique called trap-directed polling [38, pages 75–76] to offset, at least partially, the limitation of a purely synchronous system. However, SNMP traps are used primarily to indicate abnormal conditions and have to be statically defined [7], [8]. The *event* group in the RMON MIB [26] provides a method for event specification. However, the event types are predefined and thus preclude the creation of a powerful hierarchy of user defined events. Further, event relationships are not supported. Thus SNMP traps and RMON events are insufficient to convert the predominantly poll-based Internet model into a more flexible asynchronous model.

A purely event-driven system [44, pages 40–41], on the other hand, is completely asynchronous with no need for polling. This has the limitation that an event, be it important or not, has to be handled by the manager. A balance between the two would be ideal. Such a balance can be achieved by introducing an event model at an IM in the existing Internet framework. In SHAMAN, the scope of the event model is to define the basic building blocks and empower the user with the flexibility of programming network management tasks. Event correlation can be built on top of the event model using the scripting language facilities.

The addition of an event model achieves the following goals:

- provides the transition from synchronous to asynchronous architecture.
- permits effective delegation when the events to be monitored are set up remotely.
- empowers the user by permitting the creation of event hierarchies that characterize custom management tasks.

Events form the basis for the event model. An *event* is an occurrence that is caused by:

- A change in the control or data part of one or more cells
- a system related change (e.g., a timer tick, SNMP PDU receive)
- the execution of one or more cells in a spreadsheet.

An event is persistent until all the cells that are dependent upon the event have been notified of its occurrence. When an event occurs, the event id and event specific details are made available to the receiving cell.

Events can be either *basic* or *user-defined*. *Basic events* are intrinsic to the event model. These events form the basic building blocks for an event hierarchy. SNMP related events, event hierarchy related events, error events, spreadsheet configuration events, and time related events are basic events. *User-defined (or derived)* events are those events that are built using a combination of basic and other user-defined events.

For each SNMP protocol operation, a basic event is defined in the event model of SHAMAN. An *mgrget* event is generated when a manager's GetRequest is received by the IM. The cell which contains the cell-element whose OID is specified in the request is the source of this event. If variables requested in the GetRequest span multiple cells, then multiple events are generated, one for each cell containing a requested variable. A manager's GetNextRequest and GetBulkRequest result in the generation of an *mgrget* event when the manager's request is received by the IM. An *mgrset* event is similar to the *mgrget* event except that it is generated when a manager's SetRequest is received by the IM. The *agenttrap* event permits the capture of a specified Trap from an agent. All the SNMP events are enabled by default for all cells. This is to ensure that all cells support the standard SNMP interface.

The event hierarchy related events represent the minimal, expressive set needed to establish event dependencies and build event hierarchies. Event hierarchies are fundamental to this paradigm and contribute to the power of SHAMAN. A cell dependency can be expressed using one of two criteria: 1) the value of a cell or 2) the event clause in a cell. For instance, consider two cells A and B. Cell A may depend upon the value(s) in cell B or may be dependent upon the event contained in cell B. Further, in the case of value dependency, the cell A may depend upon a specific value of B or in any value change in cell B. For example, cell A may be dependent upon the value of cell B exceeding a threshold to notify a manager; or the cell A may want to wait on any change in value since the computation in cell A uses the current value in cell B. The event set must include events that can characterize the specification of these dependencies. The two events *value change* and *event occurred* capture these dependencies.

```
cell[1,1]: // Sum of counters
    on: timer(5)
    action:
        $$.1 = [1:2].1 + [3:4].1
```

Fig. 8. Example of Timed-execution Cell

A *value change* event is generated when a cell's value changes. The *value change* event carries the old and new values. These values aid in determining if a value based event condition is met or not.

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 *error* (exception) event is generated when an error is encountered during script processing. If the script execution encounters a syntax, type check, semantic or other error, then an *error* event is raised. This allows the IM to perform error processing and in some cases notify the appropriate manager.

A *timer* event is generated on every clock tick. A time value may be optionally specified 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. Optionally, a time unit can be specified (e.g., seconds, minutes), the default time unit being seconds. For example, the script in Figure 8 contains a timer event specification *timer(5)*; its event condition is then triggered every five (5) seconds resulting in the execution of its action. 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 IM from an agent for an OID that is contained in a cell. The poll event contains the new value of the variable.

C.1 Cell Script Structure

The control part of a cell consists of three parts: an initialization part, an *event-specification part* (or event expression) and an *action part*. The general structure of a cell script in SSL is:

init: $<init - statements>$
on: $<event - expression>$
action: $<SSL - Statement - block>$

Here, $<init - statements>$ refers to a set of valid SSL statements that are executed when the script is set up. The optional event expression, specified using the **on** keyword, acts as a filter if it is present. The $<event - expression>$ may specify a basic event or a user-defined event and is similar to a boolean expression. Basic and derived events can be combined using the boolean operators defined in SSL (i.e., $||$, and &&). When used in conjunction with the **on** keyword, the expression is treated as an event.

Cells in which the event specification part is present are called *event cells*. An *executable cell* is a cell that has no event specification part. The action part is a sequence of SSL statements. In an event cell, these statements are executed only if the event expression is true.

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 IM before a value is returned. An executable cell can be executed 1) on a manager GetRequest or 2) as part of another script processing. A variation on this is the *timed-execution cell* that permits periodic execution of a cell based on a time criteria. For example, a cell could be set up to sum a set of related counters every hour.

The use of derived events in an event expression makes it possible to build a hierarchy of events, all of which are based on the set of basic events supported by the event model. 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. This whole processing cycle is called an *event-processing-cycle*. The notion of an event processing cycle is used in determining the nodes visited in the event-dependency graph.

C.2 Operation of the Event Model

The primary functions of the event model include generation of events, propagation of events and processing event expressions and dependencies. An event is either generated by the system (e.g., SNMP based events, timer) or by a cell (referred to as the source cell). If there are any cells that are dependent upon the generated event (referred to as target cells), these cells should receive the event for the event dependency to be functional. These dependencies are modeled as a tree with a root node being the source event and the dependent cells being leaves of the tree. Propagation of events reduces to tree traversal, and the spreadsheet can be visualized as a collection of such trees (i.e. a forest) with each tree representing the propagation related to a specific event.

The execution of one cell may be made dependent upon events in another cell through value change and event occurred events. These dependency specifications give rise to a causality graph [45]. An event expression may be combined using the || or the && operators for which the causality graph is similar to the well known AND-OR graph [46, pages 102–106]. Thus evaluating event expressions defined in this event model reduces to AND-OR graph processing. Evaluating if a condition is true is equivalent to determining if a rule is true in the AND-OR graph. Thus the spreadsheet can be modeled as a set of AND-OR graphs with each graph representing a specific event hierarchy. In the case of SHAMAN's event model, each event forms a node in the graph. If events are combined in a cell, then the cell forms an interior node with the AND/OR combination reflected at the cell's node. The cell dependencies are also part of the event specification and thus follow the same rules as the basic events while constructing the AND-OR graph.

The definition of the scope of an event is two-fold: 1) scope with respect to event propagation and 2) scope with respect to an event expression evaluation. An event is available to the system until all the target cells have received the event. The cells that receive this event may store this event depending upon the event expression contained in the cell. Once the event is invalid from an event expression processing point of view, the event is discarded by the cell.

When an event is generated by a cell, the event is delivered to all the nodes (cells) on the event propagation tree. These cells, in turn, re-evaluate their event expressions to determine if their event conditions are met. If so, they generate the *event occurred* event and the event propagation cycle for this new event is initiated. This cycle continues until all the events triggered as a result of the original event have been forwarded to the respective target cells. Thus the occurrence of an event in the system can lead to the generation of other events and the execution of one or more cells leading to a powerful, well-knit control specification. Thus, a set of cells, each performing a simple sub-task, can cooperate to collectively perform a significant management task.

IV. Prototype Implementation of SHAMAN

A prototype implementation of SHAMAN has been developed at the University of Delaware; the current version called Version 2.0 is available at [13]. This prototype is based on the existing UCD SNMPv2 package [47]. The standard SNMP agent in the UCD package has been extended to implement SHAMAN's IM (Intermediate Manager). The extensions include support for the Spreadsheet MIB, the Spreadsheet Scripting Language (SSL), and the event model. Our prototype implementation was initially developed to run under SUNOS 4.1.3, but has since been ported to Solaris 2.6.

Figure 9 shows the software architecture of the IM and the interdependencies of the various modules that constitute the IM. Among these modules, the MIB Module, the Interpreter Module, and the Cell Module together implement the three logical components of the IM, i.e., SSMIB, SSL (interpretation) and the event model. The Polling Subsystem implements the polling of the agents. The other modules perform support functions like timer services and providing a communication interface for polling the agents.

The SNMP Communications Module deals with encoding, decoding, sending, and receiving SNMP PDUs; this module is a modified version of the UCD SNMP agent. When a manager requests an operation on the control portion of a cell (i.e., the control table), this module routes the request to the MIB module for processing. However, if the manager request is for the data portion of the cell, the request is routed to the Cell Module via the input queue. The SNMP communications module is also responsible for returning responses to the manager and for providing services to the SNMP Interface module which include encoding and sending SNMP requests to the agents being polled, and receiving and decoding the

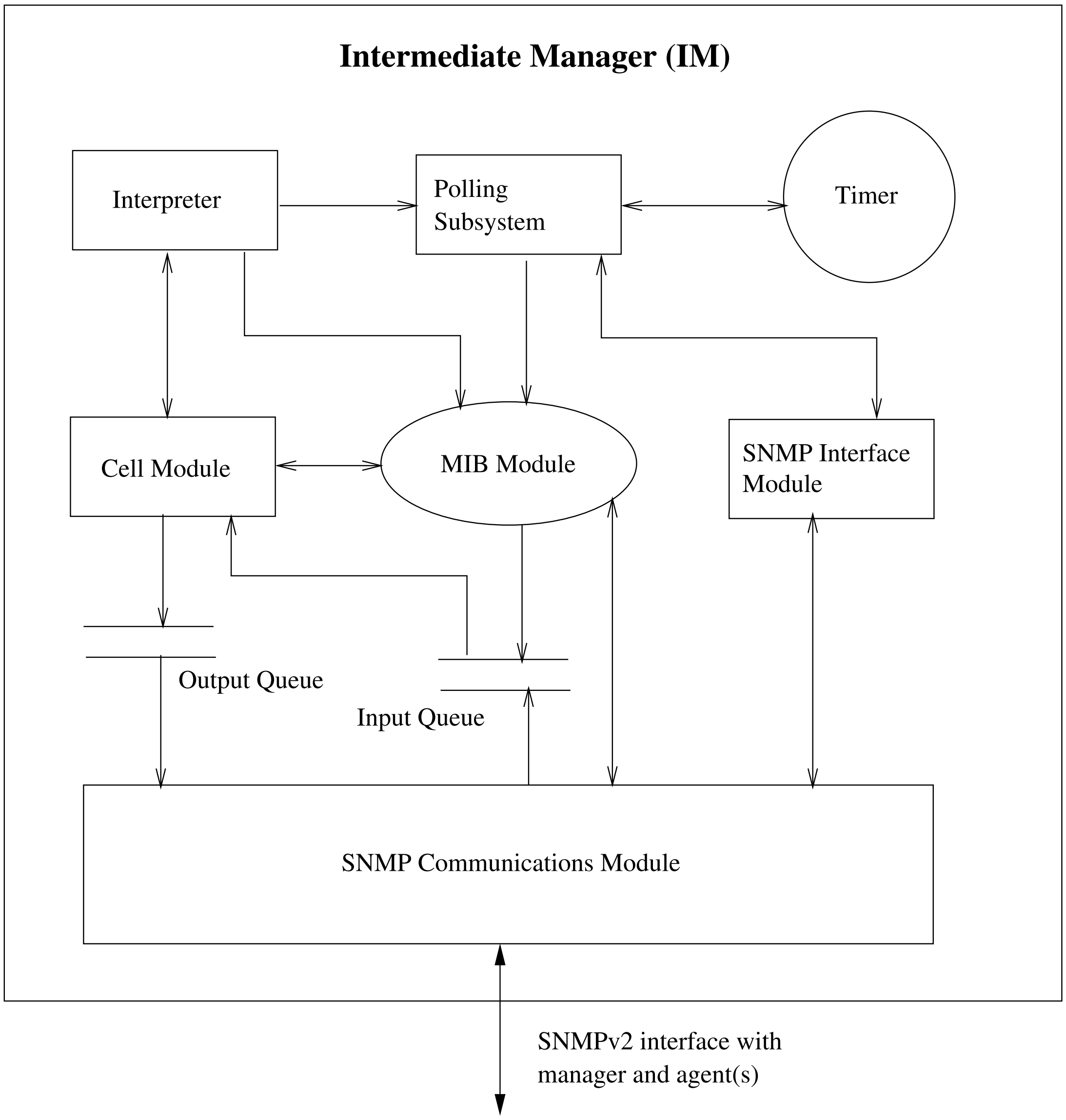

Fig. 9. Software architecture of an Intermediate Manager (IM) in SHAMAN

responses received.

The MIB Module provides an implementation of the SSMIB internal structures and controls access to it. Any module that accesses the variables in the SSMIB has to interface with this module. The MIB Module implements the SSMIB tables described in Section III-A. It supports row creation in the data and control tables; it also supports variable retrieval and modification. From the spreadsheet perspective, this module supports the retrieval of one or more values from a cell, and assignment to a single cell-element. In addition, this module also generates the events resulting from such operations.

The primary function of the Interpreter Module is to interpret the SSL (described in Section III-B) and support the SSL virtual machine. This module interfaces with the Polling Subsystem to create or delete polling entries, and with the MIB module to retrieve or modify SSMIB data table variables. The interpreter consists of two main parts: 1) a lexical analyzer built using LEX [48] and 2) a parser-cum-semantic analyzer built using YACC [48] (an LALR parser generator). A separate context and a statement counter are maintained for each script. The statement counter acts as the program counter for the SSL virtual machine. The Interpreter Module provides services that permit token analysis, parsing, type checking, execution of scripts, error detection, and reporting. Additional details can be found in [40].

The Cell Module implements the cell abstraction. To support the abstraction, this module implements interfaces for cell creation, deletion, and execution. In addition, this module manages the context of each cell and performs event dependency processing. When an SNMP PDU that accesses the data table is received by the IM, the SNMP Communications Module reads the PDU and generates a suitable internal event that is put into an event queue. The Cell Module reads the event queue and processes the events, one at a time. The Cell Module determines the target cell and thereby the SSMIB variable, based on the OID of the variable requested, and performs the requested operation. Further, the Cell Module determines the set of dependent cells that are recipients of a given event, and routes the events to those cells.

Each cell has a context associated with it. The cell context keeps track of a cell's local variables, control and data table pointers in the SSMIB for easy, quick access; in addition, the cell context contains event dependency information, queues for storing events targeting the cell, status flags, and the type of the cell. Information needed to process SNMP requests (e.g. request id) are also a part of the cell's context. When a cell is created, an associated context is created. This allows a cell to be self-contained and retain history sensitivity across invocations, if necessary. When a cell is deleted, the associated context is removed.

The Cell Module also maintains event dependency information. The event dependency information consists of a list of <*cellid,eventcode*> pairs. When an event expression is true, the action part of the cell script is executed and the appropriate event is forwarded to the cells listed in the event dependency list.

The Polling Subsystem design is modular and allows different polling strategies to be implemented without affecting the other IM modules. The Polling Subsystem consists of a polling manager, core control module, variable lookup module, polling table manager and a polling dispatcher module. The polling manager module exports interfaces to create, delete, and lookup poll variables and provides services to start polling, stop polling, and update polled variables. The core control module permits the selection of the appropriate polling strategy. Using this module, poll requests can be requeued or suspended. The value lookup module provides interfaces to create, delete and lookup polling table entries. This module also facilitates the lookup of variables being polled. The polling table manager, as the name suggests, manages the polling table maintained by the Polling Subsystem. The polling dispatcher module is primarily responsible for forwarding poll requests to the SNMP interface module. This module provides services that permit queuing, and dequeuing of poll requests, call back interface for processing poll responses, updating values in the spreadsheet, and call back service for initiating poll requests.

The Timer Module provides the timing services for the IM. It consists of a timer service routine that is invoked on every timer tick to maintain a progressive logical IM clock, an interface to the Polling Subsystem to provide a poll timer, and a generic timer management service that allows creation and deletion of timers and maintaining these timers in a sorted list.

The SNMP Interface module provides agent side communication and hides the details of SNMP from the Polling Subsystem. The SNMP interface module interacts with the Polling Subsystem on one side and the SNMP Communications Module on the other and supports interfaces that allow the modular replacement of polling algorithms to experiment with different strategies.

In addition to the Intermediate Manager implementation, SHAMAN includes a frontend GUI with a collection of development and design tools to facilitate configuration, script creation, downloading, and management. The UI has a language assistant that provides a friendly editing environment to create SSL scripts. Further, the UI permits displaying graphs that are linked to cells and plot values of cell variables over time. The GUI was built using X-Motif [49] and the X toolkit intrinsics [50]; further details of the GUI can be found in [40].

A. Spreadsheet-based Intermediate Manager Operation

We now illustrate the operation of the IM by extending the example cited in Section III-A. Recall that in this example a manager monitors the *tcpActiveOpens* counter from three nodes: sol, tweety, and stimpy. Here we extend the example by adding the clause that the manager is interested *only* in the sum of these counters and not in the individual values of the counters. The manager sets up a spreadsheet (at the IM) that contains a *PollingCell* and a *SummationCell*. The PollingCell contains the polled values of the counters on the agent nodes, sol, tweety, and stimpy. The SummationCell is dependent upon the PollingCell values, and contains a script that executes to compute the sum of the three counters when any one of them changes.

The scripts, developed using the User Interface (UI) located on the manager, are loaded into cells using SNMP SetRequests that also result in the creation of the appropriate cells. The SetRequests received by the IM are routed to the MIB Module which creates the control portion of the cell, if it doesn't already exist. The control portion of the cell (i.e., a row in the control table) is created using standard SNMPv2 row creation semantics [41, pages 288–289]. Once the control portion of the cell is created, the Cell Module creates the associated cell context. The control script is then passed to the Interpreter Module for determining the number of the data values in the data portion of the cell, and to execute the initialization part of the script. The data portion (in this case 3 counters) of the cell is created by the MIB Module. The manager is informed about the result of the SetRequest operation using an SNMP GetResponse PDU. When the script that computes the sum is loaded, the Interpreter Module uses the Cell Module to register a dependency of the SummationCell, on the values contained in the Polling Cell. This registration causes the SummationCell to be notified of changes in the PollingCell's values. At this juncture, all the control has been set up and the variables are polled at the specified frequency.

The Polling Subsystem is responsible for polling the *tcpActiveOpens* counters on sol, stimpy and tweety. The Polling Subsystem uses the services of the Timer Module to maintain a poll timer; when this timer expires, the Polling Subsystem is notified by the Timer Module. The Polling Subsystem interacts with the SNMP Interface Module to group multiple variables into a PDU (if possible); in this example, however, no grouping is possible. The poll requests (SNMP GetRequests) are forwarded to the agents. When the corresponding responses arrive at the IM, the SNMP Interface Module informs the Polling Subsystem. The MIB Module is used by the Polling Subsystem to update the appropriate row (cell-element) in the data table with the value of the polled variable. The update causes a value change event to be generated, and delivered to the SummationCell. The event originates in the MIB Module when the PollingCell is updated, and is forwarded to the SummationCell via the Cell Module. The SummationCell (Cell Module) verifies that the event received causes the event condition to become true, and uses the Interpreter Module to execute the action part of the script. The action part computes the sum of the current values of the counters and the SummationCell's data portion is updated to contain the current sum. The manager can then retrieve this value using an SNMP GetRequest. Thus, the sum of the counters gets updated without the manager's intervention and the manager can retrieve the current value as and when needed.

V. Example Application: Location Management in a Battlefield Network

We now describe an example application for SHAMAN that will highlight some important features of the SSL and the event model. The example involves location management in a battlefield network scenario with potentially thousands of mobile nodes that are controlled from command centers which need to constantly keep track of the current location of each node [14].

Consider a group of nodes that individually move on a battlefield according to the needs of the situation. Each node requires to be periodically monitored by a manager that keeps track of the current location of the node and the amounts of fuel and ammunition left. The manager may take appropriate action if these amounts fall below specified limits.

Each node has an SNMP-manageable MIB with the following variables:

- *xPosition* with the x coordinate of the current node position,
- *yPosition* with the y coordinate of the current position,
- *remFuel* indicating the amount of fuel remaining in the vehicle,
- *remAmmo* as the amount of ammunition remaining.

The total number of such nodes to be managed may be too large for a single manager to handle. Moreover, there may be distance constraints so that we may wish to have a node be managed by a manager that is located close by. Figure 10 depicts a hierarchical management solution using the SHAMAN approach that is appropriate for this situation.

We designate two Intermediate Managers, named IM1 and IM2, with management authority over nodes that are within their spheres of management as shown by the circles in the figure. Each IM periodically polls each node

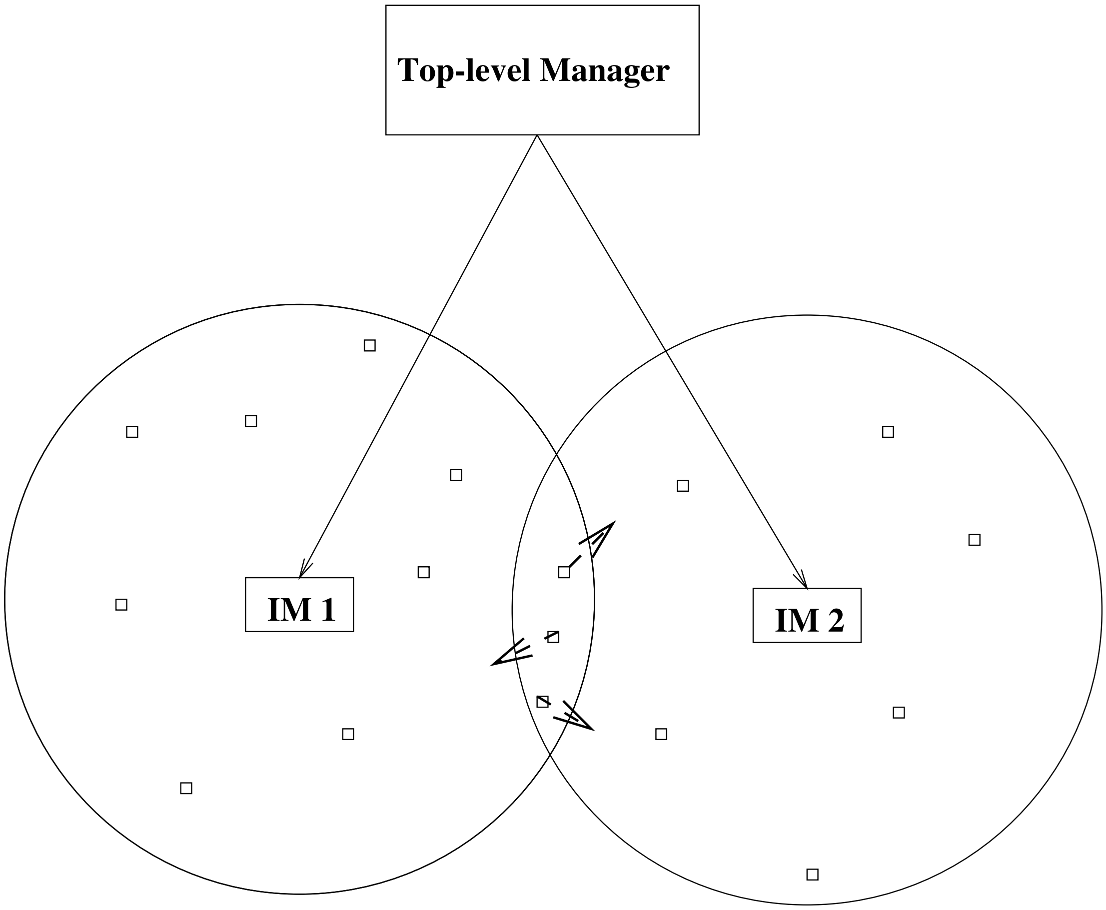

Fig. 10. Hierarchical Location Management for Mobile Nodes in a Battlefield Network

within its management domain to obtain its current variable values. If any action is required for the fuel or ammunition, then the top-level manager is informed.

As the nodes in this system move around, they may migrate from the management domain of one IM to the domain of the other. This may necessitate a "handoff" of the management authority over this node to the second manager. The need for a handoff may be detected by the IM responsible for each node. Each time a node's location is polled, it can be determined if the node has entered an intermediate zone (shown in Figure 10 as the intersection of the two management domains). If it has, and if it is rapidly moving towards the second zone, the top-level manager is informed which then initiates the handoff of the node to the second IM. This information about rapid movement can be determined from the previous and current position of the node.

Figure 11 shows the structure of the spreadsheet that contains the control for this location management application. Since the actual variables that reflect the current position are located in the respective agents, the SSL is used to set up the polling of these variables as a script in a cell that we will refer to as a *polling cell* (shown as P in Figure 11). In order to keep track of the past and present positions of the nodes, a *history cell* is used (shown as H in Figure 11) . The *history cell* stores the previous and current values of *xPosition* and *yPosition*. To determine which IM's domain the node is currently in, a *status cell* (shown as S in Figure 11) is defined. This cell contains a script that computes the current domain for the node, given the past and current location of the node. There is one cell of each type for each node being managed. In addition, dependencies must be set up to link the history cell to change when the polling cell changes. Further, the status cell needs to be linked to the history cell to perform the domain computation when the values in the history cell get updated. Once these cells are set up, as the location information gets updated, corresponding updates occur in the history cell, resulting in the recomputation

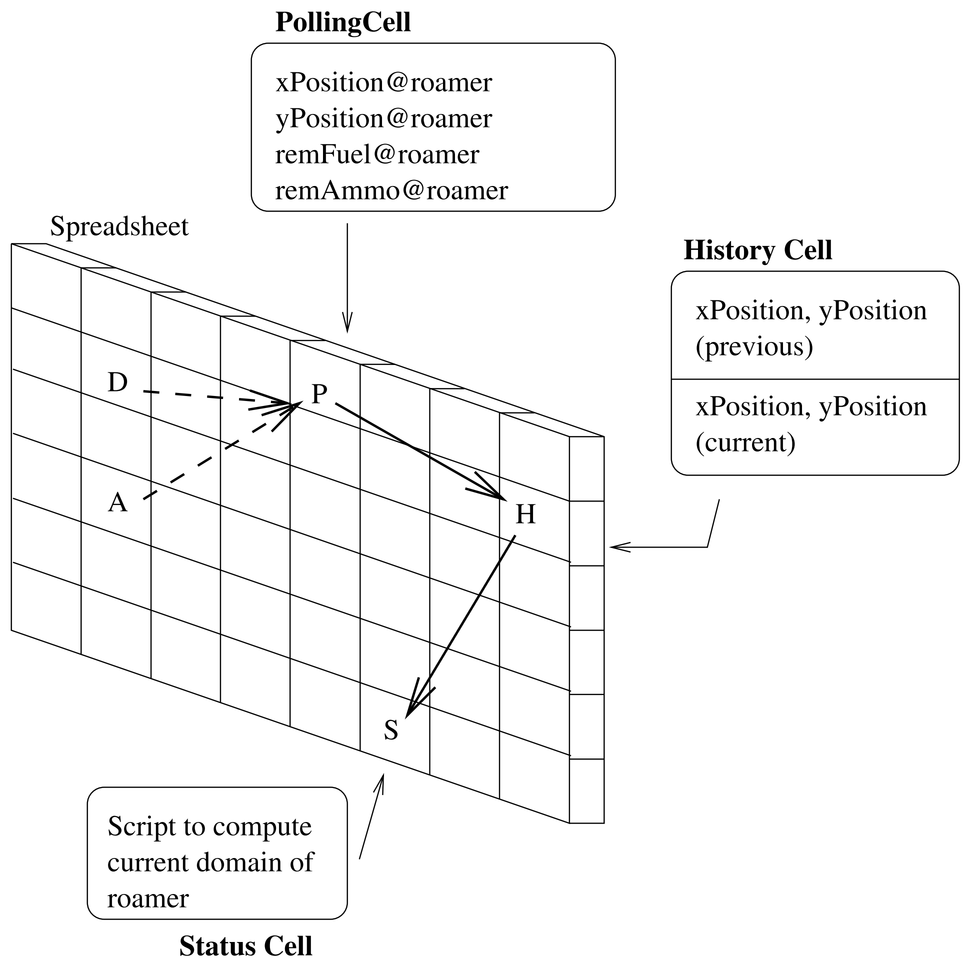

Fig. 11. Spreadsheet Structure for Location Management

of the status. A (higher-level) manager can poll the status cells to determine the current location and status of the nodes.

If a node has transitioned from one IM's domain to the other, the manager needs to perform the "handoff" operation. The manager performs this operation by informing the IM, under whose domain the node was previously located, to stop polling the node. In addition, it informs the IM under whose domain the node is currently located, to start monitoring the node. Polling is suspended when a manager executes the script contained in a *deactivation cell* (shown as D in Figure 11); polling is resumed when the script in an *activation cell* (shown as A in Figure 11) is executed.

Figure 12 shows the script set up in the polling cell for a node named *roamer*. A MIB variable is denoted in the script by its fully qualified name which includes the variable name and the hostname where the MIB is located. For instance, the MIB variable *remFuel* in the node *roamer* is referred to as *remFuel@roamer*. The values obtained from the polls are stored in the four internal values of the form $$.n. The polling cell is updated periodically through polls of the MIB variables at the agent at the default frequency configured at the IM.

```
cell[2,0]:
   label: PollingCell;
   init:  poll xPosition@roamer;    // sets $$.1
          poll yPosition@roamer;    // sets $$.2
          poll remFuel@roamer;      // sets $$.3
          poll remAmmo@roamer;      // sets $$.4
```

Fig. 12. Location Management: Polling Cell Script

The history cell for the node *roamer* is set up using the script shown in Figure 13. The event specification [PollingCell].1 && [PollingCell].2 causes the action part

```
cell[4,0]:
  label:  HistoryCell;
  init:   $$.1 = 0; $$.2 = 0; $$.3 = 0; $$.4 = 0;

  on:     [PollingCell].1 && [PollingCell].2;
  action:
    $$.1 = $$.3;              // save old x value
    $$.2 = $$.4;              // save old y value
    $$.3 = [PollingCell].1;   // copy new x value
    $$.4 = [PollingCell].2;   // copy new y value
```

Fig. 13. Location Management: History Cell Script

of this cell (the history cell) to be executed whenever the values of *xPosition* and *yPosition* in the polling cell change. Further, this script copies the old values of the x and y positions to be saved in local variables $$.1 and $$.2, and the latest polled values to be copied from the polling cell into local variables $$.3 and $$.4. Thus, this cell maintains two sets of x and y coordinates, the current set and the previous set. This information can be used to compute the velocity of the node, or its direction of movement.

The status cell (shown in Figure 14) executes each time there is a change in the values of HistoryCell. The computation performed in the action part of this cell is at the heart of the location management function executed in the IM that is responsible for managing a given node. The value of the cell, $$.1, contains the current status for this node, with 1 indicating that IM1 is responsible for managing it, 2 indicating that IM2 is responsible, while 3 indicates that the node has entered a transition zone (in the overlapping part of the management domains). When the node enters the transition zone and is traveling fast (indicated by a large change in its x-position), management responsibility is immediately transferred by changing the status to 2. Otherwise, we wait until the node has reached the end of the transition zone. If the status of the node changes to 2, the top-level manager may be informed by an SNMP Trap (not shown here) so that appropriate changes may be effected in the spreadsheets of the two IMs. For instance, the polling of this node by IM1 may be disabled while polling by IM2 may be enabled.

As mentioned before, the polling is suspended using a deactivation cell and resumed using an activation cell. As part of the activation, the status of the node is updated to reflect the IM on which the cell is being activated. The activation cell shown is for IM1 and thus shows the status being updated to 1. For IM2, the corresponding status will be 2. The two cells shown in Figure 15 illustrate these operations.

The top-level manager can get an Intermediate Manager to either start or stop the polling of node *roamer* by executing the appropriate cell above.

In addition to the cells described thus far, additional cells are used to define constants for the application. Recall that in the location management example the domains of the two Intermediate Managers are defined as circles. Two cells can be set up, one for each IM, to contain the constants that define the circular area. Each cell contains three values: the x and y coordinates of the center, and the radius of each circle. In Figure 16, cells [1,1] and [1,2] contain the three values for IM1 and IM2 respectively. Cell [1,3] defines the threshold (in the X-direction) above which the node is assumed to be moving fast. If this threshold is exceeded when a node is in the overlapping area of the two IMs, then a "handoff" is initiated.

The location management application described here has been implemented in SHAMAN and shown as a demo in some of the ARL/ATIRP Annual Symposiums; it is also available from the SHAMAN web page [13]. The power of SHAMAN is evident from this location management example. Delegation is achieved by downloading the scripts to the IM and letting the IM perform the location tracking task; the concept of MIB views has been demonstrated using the polling cells where only the relevant variables are selected for monitoring from the agents; event hierarchy is demonstrated using the dependencies between the polling, history and status cells. Many features of SSL are illustrated using the scripts contained in the cells.

VI. Topology Reconfiguration of Tactical Internets with SHAMAN

We now describe a second application of SHAMAN to perform topology reconfiguration in mobile wireless tactical battlefield internets. Such a battlefield environment is characterized by high degree of mobility of nodes, sudden appearance and disappearance of network elements (such as routers, switches, links, etc.). It is thus important for automatic configuration management of such networks that the appropriate network connectivity be regenerated after a change has occurred in the underlying

```
cell [6,9]:            // status of roamer
    label:  NodeStatus;
    init:   $$.1 = 1;                       // initially IM1's domain
    on:     [HistoryCell].3 && [HistoryCell].4;
    action:
            $5 = 0; $6 = 0;                 // initialize local variables
            // Using IM1 circle's center, compute the squares of the distances in the
            // x and y directions.
            $1 = ([IM1Domain].1- [HistoryCell].3) ∗ ([IM1Domain].1 - [HistoryCell].3);
            $2 = ([IM1Domain].2 - [HistoryCell].4) ∗ ([IM1Domain].2 - [HistoryCell].4);
            // Using IM2 circle's center, compute the squares of the distances
            $3 = ([IM2Domain].1 - [HistoryCell].3) ∗ ([IM2Domain].1 - [HistoryCell].3);
            $4 = ([IM2Domain].2 - [HistoryCell].4) ∗ ([IM2Domain].2 - [HistoryCell].4);

            if  ( $1 + $2 < [IM1Domain].3 ∗ [IM1Domain].3 )  // in IM1's domain?
            then  $5 = 1;  endif;                            // yes.
            if  ( $3 + $4 < [IM2Domain].3 ∗ [IM2Domain].3 ) // in IM2's domain?
            then  $6 = 1;  endif;                            // yes.
            if  ( $5 == 1 && $6 == 1 )    // is current point in both domains?
            then
                // check with the previous x position, if the delta is greater than the
                // threshold specified, it means that the node is traveling fast.
                // Based on the direction, adjust the status.
                if  ( [HistoryCell].1 > [HistoryCell].3 )        // prev x > curr x
                then          // moving right to left
                    if  ( ([HistoryCell].1 - [HistoryCell].3) > [SpeedThreshold].1 )
                    then  $$.1 = 1;         // threshold exceeded, status = IM1's domain
                    else  $$.1 = 3;         // no, still in common territory
                    endif;
                else          // previous x pos ≤ current x pos
                    if  ( ([HistoryCell].3 - [HistoryCell].1) > [SpeedThreshold].1 )
                    then  $$.1 = 2;         // threshold exceeded, status = IM2's domain
                    else  $$.1 = 3;  endif;         // no, still in common territory
                endif;
            else
                if  ( $5 == 1 )             // current location purely in IM1's domain
                then  $$.1 = 1;             // status = IM1's domain
                else  $$.1 = 2;  endif;      // status = IM2's domain
            endif;
```

Fig. 14. Location Management: Node Status Cell Script

network. In other words, the management system should respond and adapt to the changing network conditions.

An algorithm for such adaptive reconfiguration of tactical internets has been developed by Telcordia [51]. We first briefly summarise this algorithm and then outline its implementation in SHAMAN.

The algorithm uses a semi-formal approach with heuristics for network topology generation that produce good, though not optimal, results, with the advantages that the algorithm is fast and easy to implement. Two constraints are imposed on the generated connectivity: (a) Every node (either a router within a subnet or a subnet) in the network must be able to communicate with every other node via at least one path that is within H hops, where Hop Count H is a parameter, and (b) Every communicating node pair has more than one possible route through the underlying network. Constraint (a) ensures quick dissemination of the battlefield information while constraint (b) ensures robustness/service survivability, since the presence of multiple paths helps find alternate paths for communication in the event of a failure along a given communication path. In addition to the above two con-

```
cell[3,0]:      // Manager executes this cell
    label:  DisableNode;
    action:
        deactivate [PollingCell];

cell[3,1]: // Manager executes this cell
    label:  EnableNode;
    action:
    [NodeStatus].1 = 1;
        activate [PollingCell];
```

Fig. 15. Location Management: Activate and Deactivate scripts

```
cell[1,1]:
    label:  IM1Domain;
    init:   $$.1 = 100; $$.2 = 100; $$.3 = 350;

cell[1,2]:
    label:  IM2Domain;
    init:   $$.1 = 700; $$.2 = 100; $$.3 = 350;

cell[1,3]:     // Defines the threshold to determine
               //if status change is necessary
    label:  SpeedThreshold;
    init:   $$.1 = 50;
```

Fig. 16. Location Management: Cells containing constants

straints, there is also a constraint on the degree of each node (Degree Bound D) that results from physical limitations on the number of interfaces at each node.

The approach taken by this algorithm consists of two steps. First, the underlying network is partitioned into subnets in such a way that the total links required to realize a fully meshed interconnection within each subnet and among the subnets is kept as small as possible. To do this, an expression is derived for the total number of network links (L) for the above scenario, and a minimization is performed for L with respect to I, the number of nodes within a partition. Since the value obtained for I may be a real number, rounding operations may be required to get the closest integer value for I that corresponds to the minimum L. The actual partitioning of nodes can then be done based on knowledge about groups of nodes that are expected to have the highest traffic patterns among them.

After the network is partitioned into subnets, the second step applies a heuristic to generate connectivity within each subnet and across the subnets subject to the given constraints. The heuristics are based on using the well-known Dijkstra's shortest-path algorithm to compute the shortest path tree from a given node to all other nodes within the network. This shortest path algorithm is modified to accommodate the constraints that are imposed on the configuration by the application. Since the actual number of network links is determined based on this heuristic, the partition based on the minimization operations discussed in the earlier paragraph will not be strictly optimal. However, the approach as a semi-formal approach is expected to yield reasonable (though not optimal) results.

Since the connectivity is generated independently for each subnet, this task can be delegated to an Intermediate Manager (IM) for each subnet, such as those deployed in SHAMAN. These IMs can perform the computations needed to generate the connectivity and also the operations required for reconfiguring the routers within the subnet to conform to the new connectivity. The connectivity computation may be triggered either explicitly by the top-level manager, or by a change in some of the constraint parameters (such as Hop Count H or Degree Bound D), or by changes in node positions as monitored periodically by the IM.

This algorithm has been implemented in SHAMAN and demonstrated at the ARL/ATIRP Annual Conference in March 2001. It is also a part of the joint demo that has been designed for the entire task on Adaptive Network Management to showcase the technologies developed by all the partners in this task. Since the implementation is quite complex, we will not present the SHAMAN scripts here as we did for the Location Management application, but only outline the overall structure of the implementation.

To implement this algorithm in SHAMAN, we use a structure of cells in the spreadsheet which make it easy to modularize the different logical components of the algorithm. The first column of cells are used for external interaction from the SHAMAN entity, i.e., for communication with the top-level manager. This column includes a cell used as the trigger for topology generation. The manager will do an SNMPGet operation on this cell to command the SHAMAN IM to begin topology reconfiguration when it is necessary to do so. Another cell is used to store the generated topology; this topology is then set into an agent MIB that is used to drive the input to the directory service system from where it is accessible to the manager.

A second column of cells stores global parameters and data. These include the number of nodes in the network, the node addresses, the maximum limit on nodal connectivity (Degree Bound D), and the limit on the maximum length of the shortest path between two nodes (Hop Count H). The third column contains local structures to be used by the various components of the algorithm. These are the node positions (which are polled from the agents in the nodes), the path cost matrix, the direct link costs, and the predecessor of each node in the shortest path tree.

The remaining columns and cells in SHAMAN's spreadsheet are used for the code corresponding to the algorithm's components. The first of these is the Chain generation function, which initially generates chains between the nodes that are as long as possible without violating the maximum length constraints. The second is Dijkstra's algorithm which finds the shortest paths from each node to every other node. The third component is a function to create shorter paths whenever the algorithm has paths that are unacceptably long. This is done by adding direct links between the nodes so that path lengths may be reduced. The final component is a function to perform checks on the nodal degree constraint.

In the process of implementing this algorithm, a number of features in SHAMAN's Spreadsheet Scripting Language (SSL) were found to be extremely useful. These include the storage of two-dimensional tables in cells, since the adjacency matrix corresponding to the network connectivity can be stored and operated on as a table in a cell. Dynamic specification of cell names allows a cell to be specified dynamically within an expression by using a variable. This feature permits flexibility in applying an operation to a number of cells by putting the operation within a loop and changing the variable value in each iteration to refer to the desired cell. Dynamic specification of OIDs and hostnames permits a MIB variable to be accessed in SSL by using variables to specify its OID and the hostname of the node where it resides. Finally, the foreachrow and foreachcol statements permit easy looping over a two-dimensional structure or table stored in a cell without explicit or prior knowledge of the size of the table. This is particularly useful when the number of nodes in an internet changes dynamically.

VII. Conclusions

This paper has described SHAMAN, a hierarchical scripting framework that extends the existing Internet management framework to support delegation and user customization of control and data. We achieve this using a novel spreadsheet-based MIB in an Intermediate Manager that allows management tasks to be viewed as a set of scripts in related, cooperating cells in a spreadsheet. We described the different components of SHAMAN including the Spreadsheet MIB which acts as a repository for scripts and data, the Spreadsheet Scripting Language that integrates well with the information model of the Internet management framework, and an event model that supports asynchronous processing.

The concept of hierarchical management using SHAMAN is well-suited to the management of tactical battlefield networks because these networks require a scalable but flexible management strategy because of the large number of nodes and the high degree of mobility. We have demonstrated the utility of SHAMAN through the two example applications of location management and topology reconfiguration in tactical battlefield networks. A prototype implementation that includes these two applications as demos is available from the SHAMAN web page.

Acknowledgements

We wish to acknowledge the contributions of the following people to this effort: former University of Delaware students Pramod Kalyanasundaram, Chris Sherwin, and Saveen Pakala; current student Prashanth Kokati; and Latha Kant of Telcordia.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

REFERENCES

[1] 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.

[2] P. Kalyanasundaram, A.S. Sethi, C. Sherwin, and

D. Zhu. A Spreadsheet-based Scripting Environment for SNMP. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 752–765. Chapman and Hall, London, 1997.

[3] A.S. Sethi, P. Kalyanasundaram, C. Sherwin, and D. Zhu. A Hierarchical Management Framework for Battlefield Network Management. In *To appear in Proceedings of MILCOM '97, IEEE Military Communications Conference*, Monterey, CA, November 1997.

[4] A.S. Sethi, P. Kalyanasundaram, C. Sherwin, and D. Zhu. A Spreadsheet-Based SNMP Scripting Environment for Battlefield Network Management. In *Proceedings of the First ARL/ATIRP Annual Conference*, pages 251–256, College Park, MD, January 1997.

[5] A.S. Sethi, P. Kalyanasundaram, C.M. Sherwin, and D. Zhu. Battlefield Applications of Hierarchical Management with SHAMAN. In *Proceedings of the Second ARL/ATIRP Annual Conference*, pages 235–240, College Park, MD, February 1998.

[6] A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors. *Integrated Network Management IV*. Chapman and Hall, London, 1995.

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

[8] 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.

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

[10] B. Wijnen, D. Harrington, and R. Presuhn. *An Architecture for Describing SNMP Management Frameworks (RFC 2571)*, April 1999.

[11] 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.

[12] 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.

[13] SHAMAN web page URL http://www.cis.udel.edu/$\sim$shaman.

[14] A.S. Sethi, P. Kalyanasundaram, C.M. Sherwin, D. Zhu, and S. Pakala. Battlefield location management applications of shaman. In *MILCOM '98, IEEE Military Communications Conference*, Boston, MA, November 1998.

[15] A.S. Sethi, D. Zhu, Y. Zong, D.-P. Hsing, and L. Kant. Application of the SHAMAN Management System to Battlefield Network Configuration Management. In *Proceedings of the Fourth ARL/ATIRP Annual Conference*, pages 187–191, College Park, MD, March 2000.

[16] A.S. Sethi, K. Sivakumar, V. Hnatyshin, M. Humphrey, and B. Rivera. Integrating SHAMAN Management Applications with Battlefield OPNET Simulations. In *Proceedings of the Fourth ARL/ATIRP Annual Conference*, pages 319–324, College Park, MD, March 2000.

[17] A.S. Sethi, D. Zhu, V. Hnatyshin, P. Kokati, D.-P. Hsing, and L. Kant. Application of the SHAMAN Management System to Configuration Management of Tactical Internets. In *Proceedings of the Fifth ARL/ATIRP Annual Conference*, College Park, MD, March 2001.

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

[19] M. Trommer and R. Konopka. Distributed Network Management with Rule-Based Managing Agents. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 730–741. Chapman and Hall, London, 1997.

[20] S. Waldbusser, B. Stewart, A. Bierman, and M. Greene. *Distributed Management Framework*. Work in progress (Internet Draft: draft-ietf-disman-framework-02.txt), August 1998.

[21] J.A. George and L.E. Schlecht. The NAS Hierarchical Network Management System. In H.G. Hegering and Y. Yemini, editors, *Integrated Network Management III*, pages 301–312. North Holland, Amsterdam, 1993.

[22] G. Grimes and B.P. Adley. Intelligent Agents for Network Fault Diagnosis and Testing. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 232–244. Chapman and Hall, London, 1997.

[23] J. Schoenwaelder. *Letter in DISMAN Mailing List*, 1996.

[24] B. Stewart. *Event MIB*. Work in progress (Internet Draft: draft-ietf-disman-framework-07.txt), June 1999.

[25] D.J. Hughes and Z. Wu. Minerva: An Event Based Model for Extensible Network Management. In *Proc. INET'93 International Networking Conference of the Internet Society*, San Francisco, CA, August 1993.

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

[27] M. Erlinger. RMON - From Concept to Specification. In H.G. Hegering and Y. Yemini, editors, *Integrated Network Management III*, pages 73–80. North Holland, Amsterdam, 1993.

[28] T.K. Apostolopoulos and V.C. Daskalou. Network Management Services using a Temporal Information Model. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 638–649. Chapman and Hall, London, 1997.

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

[30] K. Marzullo, R. Cooper, M.D. Wood, and K.P. Birman. Tools for Distributed Application Management. *IEEE Computer Magazine*, pages 42–51, August 1991.

[31] B. Mohr. SIMPLE: A Performance Evaluation Tool Environment for Parallel and Distributed Systems. In *Proc. 2nd European Distributed Memory Computing Conference*, pages 80–89, Munich, Germany, April 1991.

[32] 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.

[33] International Standards Organization. *ISO/IEC 10164-4, Information processing systems - Open Systems Interconnection - Systems Management: Alarm Report Management Function*, 1992.

[34] A. Mayer, S. Kliger, D. Ohsie, and S. Yemini. Event Modeling with the MODEL Language. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 625–637. Chapman and Hall, London, 1997.

[35] G. Jakobson and M. Weissman. Real-time Telecommunication Network Management: Extending Event Correlation with Temporal Constraints. In A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, *Integrated Network Management IV*, pages 139–150. Chapman and Hall, London, 1995.

[36] P. Kalyanasundaram, A.S. Sethi, C. Sherwin, and D. Zhu. A Spreadsheet-Based Scripting Environment for SNMP. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 752–765. Chapman and Hall, London, 1997.

[37] N. Vassila, G. Pavlou, and G. Knight. Active Objects in TMN. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 139–150. Chapman and Hall, London, 1997.

[38] M. T. Rose. *The Simple Book: An Introduction to Internet Management*. Prentice Hall, Englewood Cliffs, NJ, second edition, 1994.

[39] P. Dini and R. Boutaba. Deriving Variable Polling Frequency Policies for Pro-active Management in Networks and Distributed systems. In A. Lazar, R. Saracco, and R. Stadler, editors, *Integrated Network Management V*, pages 541–552. Chapman and Hall, London, 1997.

[40] C.M. Sherwin. Issues in the Design and Implementation of Spreadsheet-Based Network Management Agent, 1997. BS Thesis, University of Delaware, Newark, DE.

[41] M. Hein and D. Griffiths. *SNMP Version 1 and 2: Simple Network Management Protocol, Theory and Practice*. International Thomson Computer Press, London, UK, 1995.

[42] P. Kalyanasundaram. *Proxy Architectures for Hierarchical Scripting Frameworks and Interoperability of Management Models*. PhD thesis, University of Delaware, Newark, DE, 1998.

[43] B.W. Kernighan and D.M. Ritchie. *C Programming Language, Second Edition*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[44] D.M. Chiu and R. Sudama. *Network Monitoring Explained: Design and Application*. Ellis Horwood, New York, NY, 1992.

[45] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo. A Coding Approach to Event Correlation. In A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, editors, *Integrated Network Management IV*, pages 266–277. Chapman and Hall, London, 1995.

[46] G. F. Luger and W. A. Stubblefield. *Artificial Intelligence and the Design of Expert Systems*. Benjamin

Cummings, Redwood City, CA, 1989.

[47] 1999. UCD SNMP Agent web page URL http://www.ece.ucdavis.edu/ucd-snmp.

[48] J. Levine, T. Mason, and D. Brown. *lex & yacc*. O'Reilly & Associates, Sebastopol, CA, 1992.

[49] D. Heller, P. Ferguson, and D. Brennan. *Volume 6A: Motif Programming Manual*. O'Reilly & Associates, Sebastopol, CA, 1992.

[50] A. Nye and T. O'Reilly. *Volume 4M: X Toolkit Intrinsics Programming Manual*. O'Reilly & Associates, Sebastopol, CA, 1992.

[51] L. Kant, C.-H. Zhu, D.-P. Hsing, and M. Lee. An Adaptive Configuration Management Architecture Design for the Army's Networks. In *Proceedings of the Fourth ARL/ATIRP Annual Conference*, pages 223–227, College Park, MD, March 2000.