# BATTLEFIELD APPLICATIONS OF HIERARCHICAL MANAGEMENT WITH SHAMAN

Adarshpal S. Sethi
Pramod Kalyanasundaram
Christopher M. Sherwin
Dong Zhu

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

## ABSTRACT

*Effective management of battlefield networks requires a hierarchical network management architecture wherein managers can dynamically delegate management tasks to intermediate managers. In this paper we describe a novel framework called SHAMAN - Spreadsheet-based Hierarchical Architecture for MANagement - which extends the traditional flat SNMP management model to a hierarchical architecture. We describe in detail the scripting language and event model used by SHAMAN, and illustrate its use by an example of location management in a tactical battlefield scenario. A demo of our implementation of SHAMAN and the location management application will be shown during the ATIRP Annual Conference.*

**Keywords:** Network Management, Hierarchical Management, SNMP, Tactical Internet, Location Management.

## INTRODUCTION

A hierarchical management strategy is an effective means of managing the large and complex internetworks that are in use today [1]. The need for hierarchical management is particularly acute in tactical battlefield networks which are expected to have tens of thousands of nodes. The Force XXI Battle Command Brigade and Below (FBCB2) of the PM Appliqué in its System Requirements Review [2] has identified a hierarchical management strategy for the management of the Tactical Internet as the Army War-Fighter Experiment transitions to its expanded future system versions 2, 3, and 4. In this new management approach, management information will be summarised at each level of the hierarchy and will flow upwards through the Platoon, Company, Battalion, and Brigade levels to the ISYSCON platform. Every FBCB2 node will have the ability to activate a net manager function so that there is no single point of failure. Although currently mainly monitoring functions have been identified, the management structure will likely grow to include control functionality as well.

Unfortunately, the most popular management framework in use today, the SNMP framework (which includes both the SNMP and the SNMPv2 protocols) [3], [4], [5], only supports the flat management model. 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. The FBCB2 management strategy uses SNMP since it is based on the Internet protocol suite and is therefore limited by these weaknesses of SNMP.

While the management community in general has tried to design management strategies based on the concept of Management by Delegation (MbD) [6], [7], 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-based **H**ierarchical **A**rchitecture for **MAN**agement)

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.

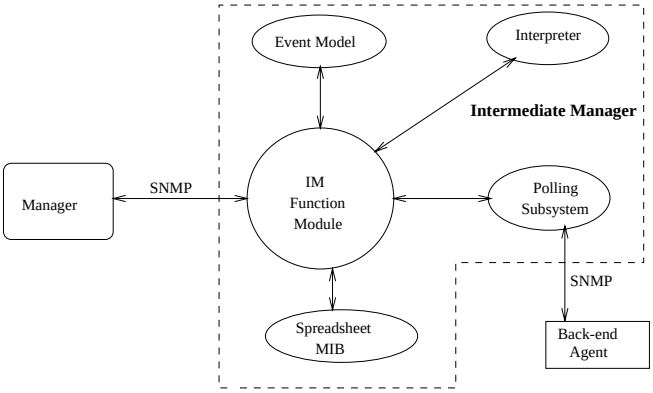


Fig. 1. Prototype Implementation of SHAMAN

that incorporates management by delegation concepts into the SNMP framework to facilitate the management of distributed systems [8], [9], [10]. 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.

In our paper at the First ATIRP Annual Conference last year [11], we described the motivation for and the basic principles behind the spreadsheet-based architecture and presented the structure of a proposed implementation (Figure 1). This figure includes the various modules that constitute the Intermediate Manager (IM) which is at the heart of the concept behind SHAMAN. Considerable progress has been made in the past year on the design of the Spreadsheet Scripting Language (SSL) and an Event Model for use in this language. We now have a prototype implementation of SHAMAN available for demonstration at the Second Annual Conference.

In this paper, we describe some of the design features of the scripting language and event model used in SHAMAN. We illustrate the use of the language with an example of a hypothetical scenario of location management for mobile nodes in a battlefield network. Consider a group of nodes that individually move on a battlefield according to the needs of the situation. Each node has an SNMP-manageable MIB with the following variables:

- *xPosition* with the x coordinate of the current node position,

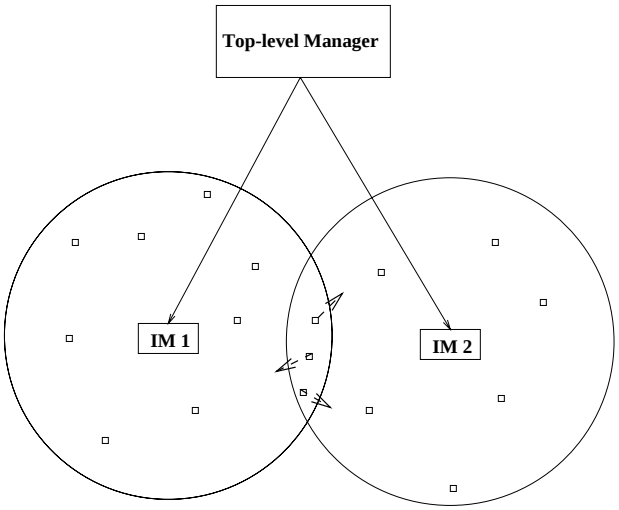


Fig. 2. Hierarchical Location Management for Mobile Nodes in a Battlefield Network

- *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.

Each node requires to be periodically monitored by a manager who keeps track of the current location of the node and the amounts of fuel and ammunition left, and which may take appropriate action if these amounts fall below specified limits.

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 2 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 communication as shown by the circles in the figure. Each IM periodically polls each node 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 the figure 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 who then initiates the handoff of the node to the second IM.

Even though the example presented here is somewhat simplistic and a real-life situation has to consider other factors such as failures of intermediate managers, it serves to illustrate the power and utility of hierarchical management. This example has been programmed into SHAMAN in the form of a demo that will be displayed during the Annual Conference.

## SPREADSHEET LANGUAGE (SSL)

A language that targets a network management environment must be able to support features that facilitate the specification of network management tasks coupled with user flexibility and expressive power [12]. This section describes the main features of the spreadsheet language (SSL) that forms an integral part of the spreadsheet paradigm and supports the development of network management scripts.

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

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

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

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

An important feature supported by the SSL is that it allows a user to set up and access multiple managed objects in a cell. A user can specify a list of objects to indicate that multiple values must be stored in the cell. Each individual value can be accessed using [s:r:c].n where $n$ represents the *nth* value in the cell. Within a given cell, the notation *$$.n* is used to refer to the *nth* value of the cell. For example, in the location management example, suppose the domains of the two Intermediate Managers are defined, for simplicity, as separated by vertical lines. The common, overlapping portion of the domains might, for instance, be delineated by $x = 380$ and $x = 500$. We might set up two cells containing these constants as follows:

*cell[1,1]:*
**label:** IM1Domain;
**init:** $$.1 = 380;

*cell[1,2]:*
**label:** IM2Domain;
**init:** $$.1 = 500;

The SSL permits a user to specify a fully qualified managed object in both symbolic and dotted decimal format. A fully qualified managed object is a com-

bination of both the OID (Object Identifier) of the managed object and the host on which it resides. For instance, suppose a mobile node in our example tactical battlefield network has the name *roamer*, then the script in a spreadsheet cell can refer to the MIB variable *remFuel* in this node as *remFuel@roamer*. This feature allows the user to identify and access any managed object in the management domain. A user can optionally specify a polling interval for a managed object. This option allows the variable to be polled at the user-specified frequency instead of a default frequency. The following cell sets up a set of polls to the MIB variables of the node *roamer*; note that the values obtained from the polls are stored in the four internal values of the form $$.n.

*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*

The SSL allows dynamic configuration of a spreadsheet using the *activate* and *deactivate* statements. Using these statements, a user can enable or disable the script contained in a cell. The following two cells illustrate these operations:

*cell[3,0]: // Manager executes this cell to stop polling*
**label:** DisableNode;
**action:**
deactivate [PollingCell];

*cell[3,1]: // Manager executes this cell to start polling*
**label:** EnableNode;
**action:**
activate [PollingCell];

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. The script contained in a cell that is deactivated cannot be executed unless it is activated again. A cell cannot be activated or deactivated when the script in the cell is executing.

## EVENT MODEL

The SNMP framework is predominantly synchronous. The primary source of asynchronous processing is the use of traps from the agent to the manager. This section describes the event model used by SHAMAN that provides enhanced support for asynchronous processing.

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

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

Mgrget and Mgrset events are generated when Get or Set Requests are received from the manager. An eventget event is generated when, as part of event processing, a request is made for executing the script contained in a cell. A timer event is generated on every clock tick. A user can optionally specify a time value as part of the timer event specification. For example, *timer(5)* implies that the event condition will be triggered every 5 seconds. This feature is useful to perform the periodic, repetitive tasks that are typical in network management applications. A poll event is generated when a poll response is received by the IM for an OID that is contained in a cell. The poll event contains the new value of the variable.

To support the event model, all cells in a spreadsheet can support both event generation and event receipt. However, cells are in general divided into two categories: *event-based cells* and *executable cells*. The control part of an event-based cell is composed of

two parts: an implicit or explicit *event-specification part* (or event expression) and an *action part*. The event-specification part acts as a filter that looks for one or more events to have occurred before executing the action part of the cell. The general structure of such a cell is:

```
on: <event_expression> ';'
action: <SSL_Statement_block>
```

where <**event_expression**> specifies the event that triggers execution of the block of SSL statements contained in the action part. The event expression could be a basic event or a derived event and is similar to a boolean expression. Derived events are obtained by combining basic or other derived events using the boolean operators defined in SSL (i.e., ||, &&, !).

To continue our location management example, the cell below contains an event specification [PollingCell] that causes the action part of this cell to be executed whenever the values in PollingCell change. Recall that PollingCell is the label of cell[2,0] which was set up to poll the MIB variables of node *roamer*.

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

**on:** [PollingCell];
**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*

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. In an executable cell, the mgrget and eventget basic events are automatically enabled. Thus, a cell containing executable SSL statements could be executed both by manager request and as part of event processing. A variation on this is the *timed-execution cell* that permits a user to set up a periodic execution of a cell based on some time criteria. Cells [3,0] and [3,1] described in the previous section are examples of executable cells.

Below is the final cell of the location management example that 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, if it 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.

*cell[5,0]: // status of node roamer*
**label:** NodeStatus;
**init:** $$.1 = 1; *// initially IM1's domain*

**on:** [HistoryCell];
**action:**
if ([HistoryCell].3 > [IM2Domain].1)
then
$$.1 = 2; *//change to IM2's domain*
else
if (($$.1 == 3) &&
([HistoryCell].3 - [HistoryCell].1) >= 100)
then *// traveling fast towards IM2's domain!*
$$.1 = 2;
else
if (([HistoryCell].3 > [IM1Domain].1)
then
$$.1 = 3; *//enter transition zone*
endif;
endif;
endif;

## CONCLUSIONS

In conclusion, we have presented a description of the scripting language and event model used for hierarchical management by SHAMAN, and have explored the application of this framework to the problem of location management in battlefield networks. This is a powerful framework that allows delegation of routine management tasks to intermediate managers and relieves the top-level manager of these responsibilities.

A prototype implementation of SHAMAN is currently in progress. The implementation includes a graphical user interface that can be used by a manager to construct, load and execute scripts in the Intermediate Manager. The location management application has been implemented on this prototype and is available for demonstration during the Conference. We will be working during the next year to complete the prototype implementation and explore the location management application in more detail. We will particularly try to include more realistic battleground features into this application. During the subsequent years, we plan to move our implementation to the ARL Testbed to assess its performance in an environment with real mobile nodes.

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

[2] *Force XXI Battle Command Brigade and Below (FBCB2), PM Appliqué, System Requirements Review II*, February 1997.

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

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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