APPLICATION OF THE SHAMAN MANAGEMENT SYSTEM TO BATTLEFIELD NETWORK CONFIGURATION MANAGEMENT

Adarshpal S. Sethi   Dong Zhu   Yuan Zong
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716
{sethi, dzhu, zong}@cis.udel.edu

Deh-Phone Hsing   Latha Kant
Telcordia Technologies, Inc.
445 South Street, Morristown, NJ 07960
{dhsing, lkant}@telcordia.com

ABSTRACT

SHAMAN (Spreadsheet-based Hierarchical Architecture for MANagement) is a novel management framework developed at the University of Delaware; it extends the traditional flat SNMP management model to a hierarchical architecture. Previous papers at the ARL/ATIRP Annual Conferences have described the framework, its prototype implementation, and a location management application for tactical battlefields. Telcordia has developed an adaptive configuration management application for tactical networks which is an ideal candidate for implementation in the SHAMAN framework. This papers presents an overview of SHAMAN and the configuration management application, describes enhancements to the SHAMAN framework to simplify configuration management with it, and outlines the proposed strategy for implementation.

Keywords: Network Management, Hierarchical Management, SNMP, Tactical Internet, Configuration Management.

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 (Spreadsheet-based Hierarchical Architecture for MANagement). 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 [1], [2], [3], [4]. 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.

A hierarchical management strategy is an effective means of managing the large and complex internetworks that are in use today [5]. The need for hierarchical management is particularly acute in tactical battlefield networks which are expected to have tens of thousands of nodes. Unfortunately, the most popular management framework in use today, the SNMP framework [6], [7], [8], 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. SHAMAN provides this much-needed capability to the Internet management framework.

Our work over the past year has added new features and capabilities to the hierarchical framework of SHAMAN. In particular, these include features that make it easier to use SHAMAN in peer-to-peer interactions. At the same time, work done at Telcordia has resulted in the design of an algorithm for
adaptive configuration management [9]. This paper explores how the new features of SHAMAN can assist in implementing the adaptive configuration management algorithm for the Army’s tactical battlefield networks. We start in Section 2 with a brief overview of SHAMAN. Section 3 describes Telcordia’s Adaptive Configuration Management System. Section 4 presents a summary of the new features added to SHAMAN and outlines possible strategies for implementing battlefield network configuration management application with SHAMAN’s new features. Finally, Section 5 presents the conclusions and outlines the future work.

II. Overview of SHAMAN

SHAMAN allows a manager to delegate tasks to an intermediate manager (IM) by downloading scripts expressing these tasks into a spreadsheet-like structure of the IM called the Spreadsheet MIB [10], [11], [12]. This MIB is divided into a two-dimensional structure of cells called a spreadsheet, with each cell having a control part that stores the script and a data part that contains the result of executing the script. One IM can support multiple spreadsheets.

The spreadsheet MIB implements the spreadsheets using SNMP tables. All operations on the spreadsheet including a manager’s downloading of scripts into the control parts and accessing the results in the data part are carried out via the Get and Set operations of the SNMP protocol. 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 returns a response to the manager that requested the cell operation.

The scripts in SHAMAN are written in a special language called the Spreadsheet Scripting Language (SSL). This interpreted language contains features that facilitate the development of procedural scripts as well as event specifications. The language includes:

- 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 prototype implementation of SHAMAN has been developed at the University of Delaware; the first version dubbed Version 1.0 containing most of the basic features of the SSL has been completed and is available on the WWW at the URL

http://www.cis.udel.edu/~shaman

Figure 1 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 three of the logical components of the IM. 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.

We have also developed a demo application of SHAMAN that illustrates its use in a hypothetical scenario of location management for mobile nodes in a battlefield network [4], [11]. This application was presented and demonstrated at previous ATIPR Annual Conferences. It uses two Intermediate Managers, each running an instance of SHAMAN, to poll and manage nodes as they move on the battlefield. When a node moves out of the domain of an IM, it initiates a handoff to the other IM via a message to the top-level manager. The tasks executed at both IMs are coded as SSL scripts and loaded into their
Spreadsheet MIBs by the top-level manager.

III. An Adaptive Configuration Management System

In this section, we provide a brief overview of the design of an adaptive configuration management system (ACMS) for the Army's tactical networks. A tactical battlefield network is characterized by a sporadically changing topology, due to the sudden appearance and disappearance of the network elements (e.g., routers, switches, links, etc.). Thus, an important component of the ACMS is the need to generate/re-generate an appropriate network connectivity after a change has occurred in the underlying network (i.e., adapt and respond to network triggers). Another important component of the ACMS is the need for a database management system that can support and facilitate the configuration management operations. More specifically, the database management system will be responsible for helping the configuration management system to locate the configuration-related information during a network configuration/re-configuration phase. Examples include information such as the routers' IDs, port details, usable frequencies, and the dynamic routing protocols details, e.g., when OSPF is used with the tactical internet, the information regarding the hello-interval, periodic flooding, router-dead-interval, etc. These configuration information may be provided by a Directory Service System (DSS), for example.

In what follows, we describe, in more details, the first component mentioned above, i.e., the network topology generation algorithm and the corresponding triggering events. We have developed a semi-formal algorithm that is sensitive to network robustness and quick information reachability constraints. We provide a brief overview in this section. Further details may be found in [9].

Broadly speaking, while many schemes exist to generate network topology the complexity of the scheme increases in direct relation to the degree of optimality required. Strictly optimal solutions may not only be too complex, but also may be intractable. On the other hand, completely ad-hoc solutions may not yield consistent results. We have therefore aimed at bridging the gap between the above two extremes and propose a semi-formal approach for network topology generation that will produce good results and that are tractable and easy to implement. Since comparison with an optimal approach may not be feasible (due to the complexity and tractability problems of the optimal approach), we shall illustrate the performance of the proposed approach via proof-of-concept simulations, which is also discussed in [9].

Our approach consists of two steps. First, we have identified a rule to partition the underlying network into subnets such 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, we derive an expression for the total number of network links (L) for the above scenario, and perform a minimization for L with respect to I, the number of nodes within a partition. Since the value 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 community usage patterns. A community usage pattern is defined as a group of nodes where traffic amongst the nodes in the group is expected to be high.

After the network is partitioned into subnets, we apply a heuristic to generate connectivity within each subnet and across the subnets with the following constraints. (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 H is a parameter. (b) Every communicating node pair has more than one possible route through the underlying network. We observe that constraint (a) ensures quick dissemination of the battlefield information and 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. Since the actual number of network links is determined based on the above heuristic, the partition based on the minimization operations discussed in the earlier paragraph will not be strictly optimal. However, our approach as a semi-formal approach is expected to yield reasonable (though not optimal) results.

Next, we make the following observations about the trigger events that invoke the above mentioned network generation algorithm. In general, broad spec-
trums of candidates exist as possible trigger events. For example, a trigger to re-compute the network topology could arise out of (a) mission re-assignment requirements, (b) projected load change requirements, or (c) real-time monitored date requirements. For categories (a) and (b), the trigger is invoked based on long-term historical data to rapidly reconfigure the network with N routers in the battlefield, where N is a parameter. In category (c) possible trigger events include sustained threshold crossing of delay or loss experienced by the applications that are using the network.

Once the adaptive network topology generation algorithm computes the new topology, the actual reconfiguration can be performed by some mid-level management entities which have the management responsibility for the affected portions of the network. The SHAMAN management entities are ideal candidates for this task. Note that these mid-level management entities may need to access configuration-related information to carry out the re-configuration operations.

IV. Configuration Management with SHAMAN

Version 1.0 of SHAMAN implemented only the basic features of the Spreadsheet Scripting Language (SSL) in order to demonstrate proof-of-concept. These features were sufficient to implement the Location Management demo, although the demo could be streamlined and made more efficient if additional features were present. However, the additional features are required if SHAMAN is to be applied to more complex situations such as configuration management. In the past year, we have undertaken the task of adding these features in the process of building SHAMAN Version 2.0:

- Revision of cell structure to allow table storage: Cells can now store two-dimensional tables or data. The notation for referencing data values in cells has been revised to allow access to any specific data item within the two-dimensional structure. The cell variables can also be used in the same way.
- Dynamic specification of cell names: Cell access can now be specified dynamically within an expression by using a variable whose value specifies the cell to be accessed. 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: A MIB variable is accessed in SSL by specifying its OID and the hostname of the node where it resides. The variable can be accessed by sending SNMP Get or Set commands to the SNMP agent at that location. SHAMAN Version 1.0 only allowed hardcoding of these names, but now the OID and hostname can be assigned to variables which can be used within the poll and other statements resulting in the access.
- Foreachrow and Foreachcol statements: These statements were added to permit looping over a two-dimensional structure or table stored in a cell without explicit or prior knowledge of the size of the table. Thus, a table can be dynamically loaded into a cell, and the above statements can be used to loop over its rows and columns respectively.
- SNMP Get and Set statements: It is now possible to do an SNMP Get operation on a MIB variable as a part of an arithmetic expression. Similarly, the value of an expression can be assigned to a MIB variable resulting in an SNMP Set operation on that variable. Both operations are synchronous and must complete before execution continues. They make it easy to hide the asynchronous nature of SNMP operations and responses within a computation.

The availability of the above features now makes it possible for SHAMAN to be used in both manager and agent roles dynamically. Thus, peer-to-peer operations are now feasible so that SHAMAN need no longer be operated only in a strictly hierarchical manner. The dynamic specification of MIB variables and of cells is very valuable for configuration management in dealing with a multitude of nodes, agents, and variables.

In applying SHAMAN to the task of dynamic configuration management, we are exploring using SHAMAN entities in various roles, including:

- Using a Configuration Manager to configure the agents. This is the conventional way of managing a system configuration, except in our case, the Configuration Manager is implemented as a SHAMAN entity so that flexible but rapid prototyping can be achieved.
• Using a Configuration Agent to store the configuration information as a MIB. This Configuration Agent could be a regular agent or implemented as a SHAMAN entity for reasons of flexibility. In this case, any entity requiring reconfiguration would access this agent and be able to perform self-configuration.
• Combining a top-level manager with a Configuration Agent in a SHAMAN entity that functions in a dual role. Here most entities would be self-configuring, but the manager could handle the more difficult situations or take care of error handling.

V. Summary and Future Work

In conclusion, we have described enhancements to the framework of the SHAMAN management architecture that adds considerable flexibility to the framework by allowing peer-to-peer operations and dual role entities. This flexibility is exploited by the Configuration Management application. We also presented an ACMS algorithm that aims at regenerating a new network topology to adapt to mission re-assignments or projected load changes. The proposed semi-formal approach has the advantages of being tractable and easy to implement, while producing reasonable results for the resulting network topology. We plan to implement this algorithm on the SHAMAN platform in the forthcoming year. We also plan to port the SHAMAN implementation to the ARL Testbed so it can be used for other management tasks and applications within the Testbed.

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


