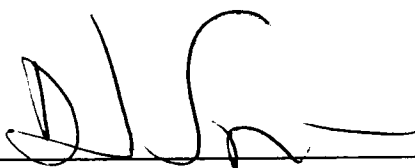


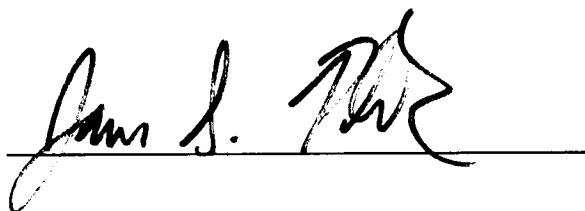
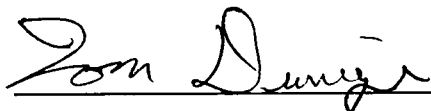
To the Graduate Council:

I am submitting herewith a thesis written by Michael R. Davison entitled "Reactive Configuration of an Internetworking Intermediate System." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science.



Dr. David Straight, Major Professor

We have read this thesis
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor and
Dean of The Graduate School

Statement of permission to use

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 6/25/94

**Reactive Configuration
of an Internetworking
Intermediate System**

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Michael R. Davison

August 1994

To my wife, Michelle, for her love and understanding,
to my parents, for their support,
and to those instructors who said, "You can!" -
Dan Descher, Ron Schwartz, and Debra Jusak.

Acknowledgments

NetEdge Incorporated, provided equipment and support which allowed this paper to be completed. I am grateful for this opportunity.

Abstract

This research describes the design, implementation, and effectiveness of a reactive approach to automatic configuration of an internetworking intermediate system. The purpose of this approach is to provide a manageable, consistent configuration engine that is capable of maintaining a desirable state. The state is modeled by a collection of abstract state objects which are manipulated by any subsystem. An event, a value transition of a state object, to a trigger value, can occur at any time such as when a network interface module is hot-swapped. When an event occurs the rule base is evaluated possibly causing configuration actions. The rule base and the group of state objects are manipulated with the Simple Network Management Protocol, allowing the use of standardized, management tools.

Contents

1	Introduction	1
1.1	Dynamic Reconfiguration	2
1.2	Reactive Configuration	3
1.3	Significance	4
1.4	Questions Addressed by this Research	5
1.5	Assumptions and Limitations	7
2	Background and Concepts	9
2.1	Traditional Configuration Techniques	9
2.1.1	Initialization	10
2.1.2	Run-Time Configuration	10
2.2	Network Management	11
2.2.1	The SNMP model	11
2.2.2	Network management Protocol	14
2.2.3	Structure of Management Information	15
2.2.4	Object Identifiers	16
2.2.5	The SNMP as a Configuration Tool	19
2.3	Reactive Configuration	19

2.3.1	What is a Reactive System?	20
2.3.2	Examples of Reactive Systems	20
3	Implementation Description	23
3.1	Design Requirements	23
3.2	SNMP Agent Architecture	24
3.3	Configuration Manager Architecture	25
3.3.1	Configuration Data	25
3.4	Summary	30
4	Results of this Research	32
4.1	Modeling System State	32
4.1.1	Hardware State Model	33
4.1.2	Current Administrative State Model	34
4.1.3	Desired Administrative State Model	35
4.2	SNMP Suitability	37
4.3	Summary	38
5	Conclusions and Recommendations	39
5.1	Conclusions	39
5.2	Recommendations	40
5.3	Summary	41
	BIBLIOGRAPHY	43
	APPENDIX	48
	A configuration MIB for the NetEdge NES 8000	49

List of Figures

2.1	The SNMP Architecture.	12
2.2	Object Identifier Prefix for the Internet	17
2.3	Object Type Formats	18

List of Tables

3.1	Example state table	28
-----	-------------------------------	----

Chapter 1

Introduction

An Intermediate System (IS) is a real-time device used to create an internetwork by connecting two or more networks. Interconnection is accomplished by receiving data frames on one of multiple network interfaces and transferring the received data to one or more other network interfaces. In most cases an IS must translate addressing information contained in the frame header in order to determine if the received frame is valid and to which networks the frame should be sent.

An IS can be a complex element of an internetwork. For example, routing a single network-layer protocol such as the Internet Protocol (IP) [Postel 81b] requires the use of several routing protocols, an understanding of an IP datagram header, the capability to quickly compute a checksum to determine if the datagram has been corrupted, and the ability to fragment the datagram into smaller datagrams if the destination network is constrained to a smaller maximum frame size. Further, an IS may route several network-layer protocols and bridge several data-link protocols.

Monitoring and controlling an IS can be accomplished in several ways: network

management frameworks, such as the framework provided by the Internet-standard Simple Network Management Protocol (SNMP) [Case 93a], remote terminal protocols, such as the Internet TELNET protocol [Postel 83], and the use of a terminal directly connected to the IS. In situations where the network or the IS itself is not functioning correctly the use of a local terminal may be required, but this approach is not very attractive since many times an IS resides in a remote location. A more desirable approach is to access the IS with a remote terminal protocol. This method is clearly more convenient than the use of a local terminal since administrators need not travel to the location of the IS. Recently, much attention has been devoted to the use of network management [Powell 91]. The frameworks provided by network management help to reduce the administrative complexity of an IS by providing a consistent, abstract model of the manageable aspects of an IS.

Configuration is an element of IS management and may be divided into two areas - initialization and synchronization. Initialization is the process of starting a subsystem and setting its initial, usually configurable, data values. Configuration synchronization is the process of insuring that configuration actions occur at the appropriate moments and that parallel operations are utilized when possible to decrease configuration times.

1.1 Dynamic Reconfiguration

Managing an IS can be a large task where configuration errors can easily cause large-scale network failure. Management complexity is intensified when external, unplanned events can cause significant changes to an IS. For example, the IS

studied in this paper, a NetEdge NES 8000¹, allows network interface modules to be inserted and removed without any preparation, an action commonly called *hot-swap*.

Typically, changing major components of an IS, such as a network interface module, requires that the IS be shut-down and manually reconfigured. Even internetworking devices which allow some form of hot-swap do not allow an interface to be configured before it is installed.

1.2 Reactive Configuration

This research describes the design, implementation, and effectiveness of a reactive approach to automatic configuration of an internetworking intermediate system. The purpose of this approach is to provide a manageable, consistent configuration engine that is capable of maintaining a desirable state. The state is modeled by a collection of abstract state objects which are manipulated by any subsystem. An event, a value transition of a state object, to a trigger value, can occur at any time such as when a network interface module is hot-swapped. When an event occurs the rule base is evaluated possibly causing configuration actions. The rule base and the group of state objects are manipulated with the Simple Network Management Protocol, allowing the use of standardized, management tools.

¹“NetEdge” and “NES 8000” are registered trademarks of NetEdge Systems Incorporated.

1.3 Significance

The reactive design of the configuration manager provides a consistent framework for the configuration of all subsystems in the IS. For example, IS cold-start is treated in the same way as interface module hot-swap by utilizing the same configuration framework and management paths. This simplifies both the management and the design of the IS.

Traditional approaches to IS reconfiguration have resulted network down-time. The configuration manager designed in conjunction with this research provides a way to reduce the amount of time an IS must be inactive during reconfiguration since a network interface module may be configured before the module is present and then the module may be inserted while the IS is operating. In comparison, traditional techniques have involved turning the IS off, inserting the new interface module, restarting the IS, and finally configuring the new interface [Cisco 92, Proteon 92].

Since the configuration manager implements a rule-based structure network managers can pro-actively manage IS faults. For example, if an interface fails and a redundant interface exists, rules could be written that would bring the inactive, redundant interface up when the primary interface fails.

The configuration manager is manipulated with SNMP. This provides a framework which allows network managers to use existing, well-understood network management tools such as the Network Management Stations (NMS) from Sun

Microsystems² and Hewlitt Packard³.

1.4 Questions Addressed by this Research

Central to the concept of reactive configuration in the NES 8000 are the notions of *physical hardware configuration*, *current administrative configuration*, and *desired administrative configuration*. The physical hardware configuration is determined by the power-on, self-test (POST) and whenever hot-swap occurs. The current and desired administrative configurations form the heart of the NES 8000 hardware configuration procedures.

The hardware state of the NES 8000 was modeled with rows in the state table which were indirectly manipulated by diagnostic tasks. The hardware state was then available to external tools via the SNMP.

1. **To what extent were the objects used to model the hardware state of the IS adequate for use in an event-driven, reactive configuration framework?**

It was anticipated that the hardware state of the IS could be adequately modeled with the Internet standard network management framework.

Modeling the current administrative state of an IS could be an extremely complex task, since the complete state of the IS is large and changes rapidly. In the config-

²Sun Microsystems is a registered trademark.

³Hewlitt Packard is a registered trademark.

uration framework developed in conjunction with this research a subset of the IS state was modeled with a group SNMP-manageable objects.

2. **To what extent does the set of IS states monitored and controlled by the configuration manager accurately model the current administrative state and do these objects provide a sufficient basis for event-driven, reactive system management?**

It was anticipated that the set of IS states modeled by the configuration manager would be a sufficient model of the current IS state.

The desired administrative state of the IS is stored in the rule table. This table contains two interesting values, a trigger value and an action to be performed when the the named state value reaches the indicated trigger value.

3. **How effectively is the desired administrative configuration modeled with the configuration rule table and to what extent does this framework allow arbitrarily complex system management actions?**

It was anticipated that the configuration manager architecture would provide sufficient generality to perform arbitrary management functions and that the rule table would be an effective way to model the desired administrative configuration.

A non-proprietary, standard interface to the configuration manager was required in order to allow remote management in a consistent way.

4. To what extent is the Internet standard network management framework suitable for remote and local management of a reactive configuration manager.

It was anticipated that, although the Internet standard network management framework limits the size and structure of management information, it would provide a sufficient mechanism.

1.5 Assumptions and Limitations

Since it is impractical to examine all issues related to this area of research, some reasonable assumptions have been made, and several limitations have been imposed.

Assumptions are any concept upon which this work builds which is not substantiated by other research.

1. It was assumed that the NetEdge network provided a "typical" environment. This environment included approximately 300 end systems interconnected with Ethernet and FDDI local area networks.

Limitations are any barrier which results in a loss of generality when utilizing the concepts developed in this, or related, research.

1. This work touches on aspects of Artificial Intelligence (AI) since rules are evaluated whenever an event occurs. In spite of this similarity to AI

applications, such as expert systems, AI was not studied in conjunction with this research.

2. The Internet standard network management framework is a critical part of the this research, and the related elements of network management are studied. However, network management is a broad topic and is not comprehensively studied in this research.

Chapter 2

Background and Concepts

When studying the run-time configuration of an IS it is useful to examine how such actions have been accomplished in the past, what techniques are prevalent currently, and what current research efforts may be useful in the future. Configuration actions have traditionally been performed by constructing, and later interpreting, some type of initialization file. Contemporary configuration methods have not progressed very far beyond this concept. Current configuration frameworks may provide a “friendly” user interface but rarely produce anything other than a simple initialization file or script. Some research efforts have, as will be discussed later in this chapter, indicated that IS configuration would be more robust and effective with the use of event-driven techniques.

2.1 Traditional Configuration Techniques

Configuration of a general computing system or of an IS has traditionally involved the use of a remote terminal protocol, such as TELNET, a command-line interface,

and an initialization file. The use of the command-line interface can be viewed as *run-time configuration* while *initialization* is normally confined to the actions modeled by an initialization file or script.

2.1.1 Initialization

Normally initialization of an IS involves entering configuration data into some type of non-volatile storage and rebooting the system. The method used to enter initialization data has varied among IS vendors. Some user-interfaces provide some type of context-sensitive help [Cisco 92, Slattery 93], while others [Proteon 92] provide a menu-based approach to guide the system administrator through the configuration process. In both cases a command-line is used to enter initialization data, and the administrator is assisted by a user interface which focuses the scope of legal commands.

2.1.2 Run-Time Configuration

As with initialization, run-time configuration has been performed with command-line and menu interfaces. Run-time configuration differs from initialization in that configuration changes are permanent and take effect when the IS is rebooted, while run-time configuration actions take effect immediately but do not become permanent.

2.2 Network Management

Internetworking allows distributed systems to be constructed from different network technologies which may be provided from various manufacturers. The multi-vendor, heterogeneous distributed environment formed in this manner does not lend itself to proprietary management solutions. Rather it requires a standard, extensible network management framework. Such an framework is the Internet standard network management framework based on SNMP.

There are several goals realized with the SNMP architecture. The first goal of the protocol is to reduce the complexity of the management agent. This is accomplished by controlling the complexity and number of management functions implemented by the agent. A second goal is that the functional paradigm be extensible, allowing new aspects of network management to be incorporated. The third goal of the SNMP architecture is that it should be independent of the architecture and mechanisms of the network elements [Case 93a].

2.2.1 The SNMP model

The SNMP architecture is composed of a collection of one or more *network management stations*, *managed nodes (agents)*, and a network management *protocol*, which is used by the management stations and agents to communicate [Rose 91b]. Management stations control network elements such as hosts, gateways, and terminal servers which have SNMP agents responsible for performing the management functions requested by the management station. This relationship is shown in Figure 2.1.

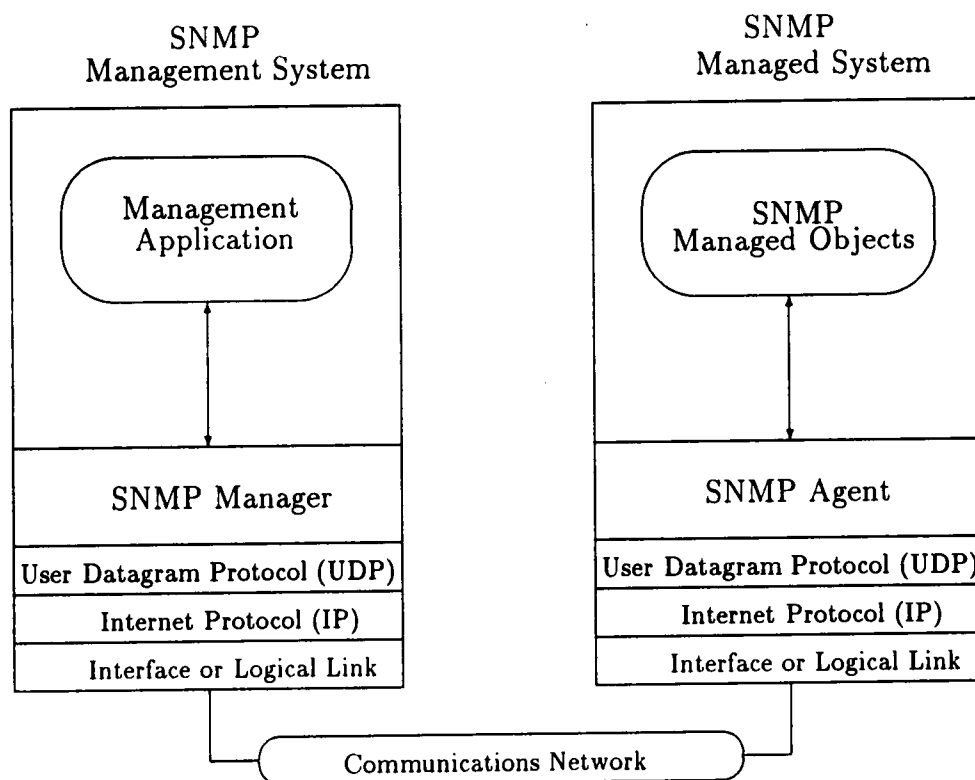


Figure 2.1: The SNMP Architecture.

Agents in Managed Nodes

Managed nodes generally fall into one of three groups: host systems, gateway systems, or intelligent network devices (such as bridges or multiplexors) [Rose 91b]. These managed nodes have network management agents that respond to requests from management stations. These agents act on network management objects, contained within the network protocols, via system specific mechanisms. On a typical UNIX node the mechanism is typically `/dev/kmem` access or `ioctl` calls [Case 90].

Significant diversity exists among the capabilities of SNMP managed nodes, and

so the SNMP management framework requires that “the impact of adding network management to managed nodes must be minimal” [Rose 91b]. This fundamental axiom is important since it allows the SNMP to be utilized on a wide range of systems. In fact, SNMP has been implemented on systems ranging in size from personal computers to mainframes [Powell 91].

Two other goals of the SNMP framework are that: first, the SNMP framework should be extensible to allow additional, possibly unforeseen properties of network management, and secondly, that the architecture should be independent of the architecture of the managed node [Case 93a].

Network Management Stations

One or more network management stations (NMS) may be used to monitor and control the network, and communicate with network management agents via an application-layer protocol, the SNMP.

A subset of the OSI presentation-layer protocol, Abstract Syntax Notation One (ASN.1) [ISO 87a, ISO 87b], is utilized to provide a “widely compatible, machine-independent mechanism that accommodates machine differences in word size and ordering, arithmetic functions, and character sets” [Case 90]. While the grammar and syntax of ASN.1 provides an abstract, human-readable syntax, it does not define the way data objects are represented in hardware, or more succinctly, it does not define the *concrete syntax*. This job is accomplished using the *Basic Encoding Rules* [ISO 87c], which define how data is transferred between hosts.

Most of the computational burden of network management is placed on the NMS

since the SNMP architecture requires that network management have minimal impact on managed nodes. This suggests that the host system for the network management station should be more powerful than the managed systems. Experience has shown that most workstations provide sufficient resources to support SNMP management stations [Rose 91b].

2.2.2 Network management Protocol

The protocol, SNMP, is utilized to transfer information between various agents and management stations. A *remote debugging* paradigm is utilized by the Internet-standard Network Management Framework. With this paradigm "Variables" are accessed on each managed node by the agent, on behalf of the NMS, via the SNMP. Nodes may be monitored by reading these variables, and may be controlled by modifying the value of these variables [Case 93d].

The simplicity offered by the SNMP remote debugging paradigm is one of the key strengths of the SNMP framework. Normal data objects can be accessed, but also, imperative commands, such as rebooting a computer, can be realized since an imperative command can be accessed by setting a variable defined for the desired purpose.

Two other actions are needed in addition to read and write operations: a *traversal* operation and a *trap* operation. A traversal operation allows the NMS to determine which variables the managed node supports. Traps allow the agent to send information asynchronously to the NMS. To summarize, the protocol operations supported by the SNMP are:

get: Retrieve specific instances of one or more managed objects from a selected managed node.

get-next: For each managed object requested the object which lexicographically follows the indicated object in the Management Information Base (MIB) is returned.

get-bulk: This operation was added with SNMP version II and retrieves up to a maximum number of objects including and following the objects requested.

set: Modifies a management object, or objects, on a selected node, to the indicated value.

trap: Agents may report exceptional conditions to one or more management stations using this mechanism.

2.2.3 Structure of Management Information

The Structure of Management Information (SMI) [Case 93b] is the second element of the Internet network management framework. It specifies how the syntax and the semantics of data objects are defined utilizing the OSI presentation-level protocol ASN.1 .

Complex data types are avoided in the SNMP framework by utilizing a subset of ASN.1. According to the OBJECT-TYPE macro defined in the SMI, the data type of a managed object must be one of the simple types listed below or one of several application-specific types.

INTEGER: A simple data type taking a cardinal number, between -2147483648 and 2147483647, as its value.

OCTET STRING: A simple data type constructed from zero or more octet. Each octet may take on any value from 0 to 255.

OBJECT IDENTIFIER: A simple data type constructed from a sequence of integers, and used to provide a unique object name.

BIT STRING: A simple data type added with version II of the SNMP which supports combining several enumerated values into a single data object.

Access of managed objects is defined by "Access", an ASN.1 class defined in the OBJECT-TYPE macro. This type defines the level of access to be one of the following values:

not-accessible: instances of the object may not be read or set. (This level of access is needed since the management profile may further restrict the level of access.)

read-only: instances of the object may be read but not set,

read-write: instances of the object may be read and set,

read-create: instances of the object may be read and created but not modified once created.

The implementation requirement for managed objects is specified by its status clause, an ASN.1 type defined in the OBJECT-TYPE macro. The status of a managed object is one of: *current*, *obsolete*, or *deprecated*.

2.2.4 Object Identifiers

Object identifiers (OID) provide a way to authoritatively name arbitrary objects, regardless of the object's semantics. A sequence of non-negative integers form an object identifier and are used to traverse a tree, like the one shown in Figure 2.2. Nodes in the tree are labeled with non-negative integers and possibly a textual description. As the tree is traversed, the naming authority may be subordinated to other organizations.

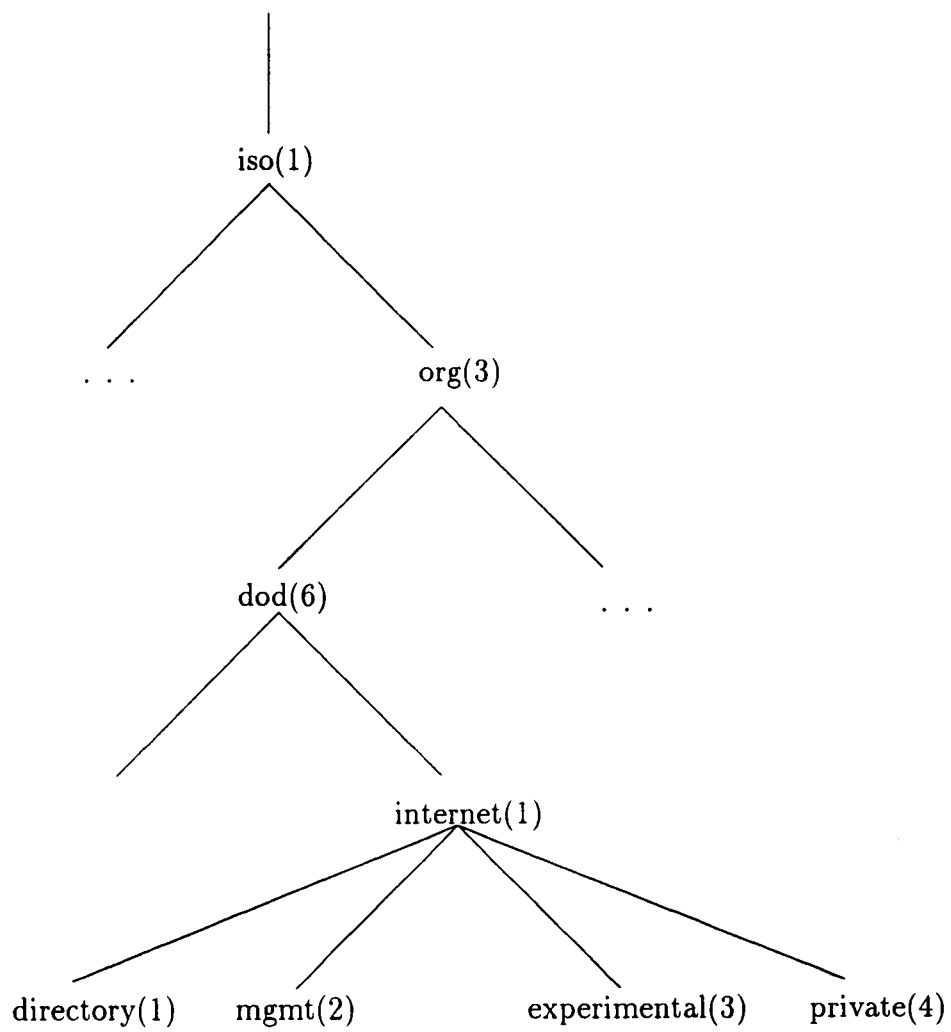


Figure 2.2: Object Identifier Prefix for the Internet

Several methods for displaying object identifiers may be used. Each format, as shown in Figure 2.3, is derived by traversing a tree, starting at the root and proceeding to the selected object. The object identifier may be listed in *number form*, *name form*, or *name and number form*. These forms are frequently combined into a *prefixed number form* where the prefix of the OID is expressed in name form and the OID suffix is expressed in number form.

Number Form:

{ 1.3.6.1.2.1.1.1 }

Name Form:

{ iso org dod internet mgmt mib system sysDescr }

Number and Name Form:

{ iso(1) org(3) dod(6) internet(1) mgmt(2) mib(1) system(1) sysDescr(1) }

Prefix Number Form:

{ system.0 }

10

Figure 2.3: Object Type Formats

An object identifier is composed of two parts: the objects type or class, and the object instance. The object type is the authoritative name for an object. In the SNMP framework, the object type would be defined in a Management Information Base (MIB) document. An example of such a MIB document may be found in the Appendix. The object instance identifies a unique instance of the indicated object type. The instance of scalar objects is zero by convention, and the instance of tabular objects typically is the value of a significant column in the table.

2.2.5 The SNMP as a Configuration Tool

Clearly, as the name suggests, the SNMP was developed to manage remote nodes over a network. It is also useful to note that one can use the SNMP to monitor and control aspects of a local IS. In other words, the management application and the agent are running on the same computing systems thus allowing standard management applications to be used for both local and remote management and utilizes the same agent software. Hence the network administrator has a consistent management tool for any IS, and the computer vendor need not implement redundant "paths" to perform management actions.

A large number of objects have been defined which can be managed with the SNMP framework. In a recently published summary of Internet standards related to network management, over 75 documents were listed [Perkins 93]. The objects defined in these documents combined with those defined in proprietary MIB documents can easily lead to hundreds of managed objects on a typical IS. Many of the managed objects in a typical IS MIB can be used to configure the IS.

2.3 Reactive Configuration

Much has been written about the use of event-driven, reactive systems in industrial, process control. In [Dulieu 87] for example, an expert system is used to monitor a detergent plant, provide operators with suggested control actions, and, in some cases, perform control actions. In this section the use of reactive systems to control real-time systems, such as an IS, are studied.

2.3.1 What is a Reactive System?

In [Harel 85] it is argued that there exist two classifications of computing systems. The first classification combines computation-oriented or data processing applications which are sometimes classified as sequential or functional applications. This type of application is reasonably well understood, but another classification involves more complex applications, in which many concurrent components, involving both hardware and software, may be present. In this paper, the terms *transformational* and *reactive* systems are proposed for these two classifications.

A transformational system accepts input, performs transformations, and produces output. On the other hand, reactive systems are continually triggered by external input. A reactive system does not produce a single output but rather is supposed to maintain a certain state.

2.3.2 Examples of Reactive Systems

The current literature is filled with applications of expert systems. One need only consult the proceedings for conferences such as "The International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems," sponsored by the University of Tennessee Space Institute and the Association for Computing Machinery, to find many examples. Some real-time expert system applications fit well with this research and are explored in this section.

Early work in the use of expert systems in industrial control outlined some of the design constraints placed on reactive real-time systems. In particular, [Khanna 86] listed the following points:

1. response time must be at least as fast as the rate of change of dynamic data,
2. such systems often require large knowledge bases as the rules/models must define the plant state under varying conditions,
3. large knowledge bases could not be built using conventional techniques, as it would require super-high processing power for an acceptable response, and
4. interactive or consultation-type approaches involving the human operators were not acceptable due, once again, to the response time.

The authors of this list go on to explain how an expert system was developed to meet these goals. While the design of the user interface is not relevant to this research, it is interesting to examine the different types of rules used to allow the user to express actions. These rule types include ways to define symbols, simulate external input, and to initialize the rule base, but the most interesting rule types are:

IF-THEN	<i>IF condition THEN action.</i>
UNCONDITIONAL	<i>action.</i>
WHENEVER	<i>WHENEVER condition THEN action.</i>

Forward chaining is used to provide timely control actions. When events occur condition-matching triggers focus on the problem area. The inference engine has also been designed to gather data, arrive at conclusions, decide on actions, and carry them out. Both forward chaining and backward chaining are used to perform the less time-sensitive function of data collection.

Building on the simple "if-then" rule construct[Taunton 86], has employed fuzzy logic to offer automated control of devices where it is difficult to describe the problem domain. This is useful since a paradox exists where, in some cases, mathematical modeling has not proven effective, yet one or two human operators can easily control the situation. In cases such as this, the rules used to control the

process are fairly subjective. Hence the term “fuzzy.” Fuzzy logic is basically a quantitative technique which expresses rules in the form

“IF the temperature is high THEN make a medium reduction to the heater.”

The value of the “medium” reduction produced by this rule would be constrained by the value of the input temperature.

Chapter 3

Implementation Description

Maintaining a desirable IS configuration can be tedious and doing so with an IS which supports hot-swap is complex. Unplanned events may occur which require rapid re-configuration to prevent excessive down-time. Furthermore, it was considered important that the configuration framework be consistent for all subsystems and that ad-hoc solutions be avoided. This chapter explores the design and implementation of a reactive configuration manager.

3.1 Design Requirements

The ability to support hot-swap is the most distinctive requirement for the NES 8000 since re-configuration may occur at any time. It was deemed unacceptable to completely re-initialize the IS each time an interface module was changed. Doing so could result in large-scale network failure for several minutes.

In order to provide a consistent management interface for the NES 8000, it was deemed necessary to support the prominent network management protocol, the

SNMP. Further, it was decided that all management actions, even those initiated within the IS itself, would utilize the same management “path.” This means that the configuration manager was required to utilize the network management path to perform configuration actions whenever possible.

Since the NES 8000 is an embedded system with relatively limited resources, the configuration manager had to be designed to utilize memory efficiently. This requirement limited the size of both data structures and code.

Finally, the configuration manager was required to perform actions upon the occurrence of an event. Actions include typical network management functions such as setting network addresses and updating routing tables, but also includes system management functions such as loading or unloading software modules.

3.2 SNMP Agent Architecture

The SNMP agent in the NES 8000 consists of an agent core and multiple subagents. The agent core is responsible for parsing an SNMP request, gathering data from subagents required to fulfill the request, and generating the appropriate SNMP response. A subagent is a task which is responsible for a portion of the Management Information Base (MIB) supported by the NES 8000.

The agent core has no knowledge of managed objects when it is initialized. Instead, subagents must register with the agent core when they are initialized. Although some objects, such as the MIB II System group [Rose 90], are provided by a standard subagent, other objects are dynamically registered and unregistered, by other

optional subagents.

3.3 Configuration Manager Architecture

The configuration manager was implemented with three distinct tasks in order to isolate complexities and to facilitate future enhancements: a system initialization task, a SNMP subagent, and a SNMP manager. The initialization task is responsible for restoring the values of any persistent configuration objects by performing a sequence of SNMP set operations.

Hardware configuration will be determined by a task external to the configuration manager, but the management interface for this configuration data will be provided by the configuration SNMP subagent. The MIB modules exported by this subagent include the *interfaces* group specified by the Internet Engineering Task Force (IETF) in [McCloghrie 88] and the NetEdge configuration MIB outlined in Appendix 5.3.

The manager task is responsible for performing SNMP set operations, as driven by the evaluation of user-defined rules, and receiving asynchronous hardware status-change messages from a task external to the configuration manager.

3.3.1 Configuration Data

Recall from Chapter 1, that central to the concept of reactive configuration in the NES 8000 system are the notions of *physical hardware configuration*, *current administrative configuration*, and *desired administrative configuration*. The physical,

or true, hardware configuration is determined by the power-on, self-test (POST) and whenever hot-swap occurs. The current and desired administrative configurations form the heart of the NES 8000 hardware configuration procedures.

The physical, current, and desired configurations are modeled with a Management Information Base (MIB) listed in the Appendix. Port objects, modeled with the *fcPortTable*, may be hot-swapped and hence require special attention. This is accomplished by allowing the *fcPortStatus* value to reflect transitory states such as *testing* or *configuring*. The port table augments the Internet standard MIB interfaces group [Rose 90].

The current administrative configuration is modeled by the state table which is outlined in the following section. The current state of a given subsystem is modeled with one or more state objects.

The Desired administrative configuration is defined in the rule table. The rule table defines what actions should be taken (desired configuration) when an event occurs (current configuration).

State Table

As was noted in Chapter 1, one of the elements of configuration is synchronization. The state table provides a generalized synchronization point for all subsystems in the NES 8000 by modeling the current administrative configuration. Each NES 8000 subsystem that needs to make its state known to other subsystems utilizes a state object to communicate its internal state. When a subsystem modifies its state, the configuration manager evaluates the rule base to determine if any actions

must be taken.

The state table is accessed with the SNMP. This allows a state object to model both the current administrative state of the subsystem and allows external entities to affect the state of a subsystem, but most importantly, this allows the use of standard, well-understood tools to used in the management of the NES 8000.

A state object is instantiated by an identifier (*fcStateIdent*) and by an identifier instance (*fcStateInstance*). Table 3.1 illustrates an example of this technique. The abstract representation of the state table is presented below using the Abstract Syntax Notation One [ISO 87a, ISO 87b].

```
StateEntry ::= SEQUENCE {  
    fcStateIdent    INTEGER,  
    fcStateInstance INTEGER,  
    fcStateValue    INTEGER  
}
```

Rule Table

Since the NES 8000 configuration manager is a reactive, or event-driven, framework the desired administrative configuration is modeled by the rule table. The rule table is the list of rules which the configuration manager uses to interpret and act upon events. An *event* is defined as the value transition of a state object to a predefined level and may occur whenever a state object is modified. The ASN.1 representation of the rule table is shown below.

fcStateIdent	fcStateInstance	fcStateValue
portState(1)	1	hssi-configuring(14)
portState(1)	4	enet-operational(23)
portState(1)	9	tr-testing(32)
portState(1)	14	fddi-operational(63)
sysMemState(2)	0	hw-operational(103)
rbMemState(3)	0	hw-operational(103)
lbMemState(4)	0	hw-operational(103)
smtState(100)	0	sys-operational(203)
spanningTreeState(101)	0	sys-testing(202)
ipRouterState(200)	0	sys-operational(203)
ipxRouterState(201)	0	sys-operational(203)
appleTalkRouterState(202)	0	sys-operational(203)
snaRouterState(203)	0	sys-invalid(201)
decNetRouterState(204)	0	sys-invalid(201)

Table 3.1: Example state table

```

RuleEntry ::= SEQUENCE {
    fcRuleIdent      INTEGER,
    fcRuleInstance   INTEGER,
    fcRuleSequence   INTEGER,
    fcRuleAction     DisplayString,
    fcRuleStatus     INTEGER
}

```

The syntax of a rule *fcRuleAction* is very simple and is presented in the following examples.

```

WHEN fcStateValue.portState.1 == enet-configuring(24)
    SET ipAdEntAddr.192.42.137.44 = 192.42.137.44,
        ifAdminStatus.1 = up(1),
        fcStateValue.portState.1 = enet-operational(23)

```

```

WHEN fcStateValue.portState.$I == hssi-invalid(11)

```

SET ifAdminStatus.\$I = testing(3)

In the first example, an Ethernet port is configured with an IP address, the interface status is changed to “up” and, finally, the port state is set to “operational.” The use of parameters is shown in the second example. Two values may be parameterized, the state object instance and value. This allows rules that should be applied to all classes of objects to be written.

Event triggers may be defined by network administrators who can add rules to the rule table. This allows administrators to configure the IS based upon possible presence of hardware. For example, a rule could be defined that called for a given network address to be associated with a particular interface when that interface became available. The interface could exist when the IS boots, be provided by hot-insertion or perhaps become available when a primary interface fails. The reactive nature of the configuration manager allows this flexibility.

All configuration actions could be performed solely as the result of defining rules. This would, however, require that SNMP subagents be running for every possible subsystem and would make configuring the IS overly complicated in some cases. It would be more efficient in some instances to load a subsystem and start its execution when it is needed. For example, it would not make sense to utilize IS resources to support routing of a particular network protocol if that protocol is not supported on the given internetwork. To remedy this problem another event action path exists.

Subsystems may register an event subroutine with the configuration manager. This subroutine will be called, in the order of registration, before any rules are evaluated.

This allows a subsystem to provide a very simple initialization routine that could start an SNMP subagent before any rules are evaluated. The event subroutine places the subsystem in a state that is configurable via SNMP. In addition to conserving IS resources, the use of event subroutines hides some of the low-level configuration activity from network administrators.

Rule instances are closely tied to instances of state objects. The values of *fcRuleIdent* and *fcRuleInstance* are equivalent to *fcStateIdent* and *fcStateInstance* respectively. When the value of the associated state object transitions to the rule trigger value, *fcRuleValue*, an event occurs, and all rules with the appropriate identifier and instance are evaluated in the order indicated by *fcRuleSequence*.

3.4 Summary

The NES 8000 configuration manager was designed with autonomous, decentralized management in mind. The state table provides an abstract model of the current administrative state without requiring the configuration manager to have any knowledge about other subsystems. This architecture allows various subsystems to be developed and changed without modification of the configuration manager.

The configuration rule table provides a way to perform arbitrary management actions upon the occurrence of an event and effectively models the desired administrative state. Further, the rule base provides the basis for future enhancements such as a expert system or perhaps the application of fuzzy logic. It is important to note that although the current rule-base evaluation is quite simplistic, these more advanced evaluation techniques could be added to the configuration man-

ager without modifying any other subsystems.

In the next chapter, this implementation will be evaluated with respect to the questions outlined in Chapter 1.

Chapter 4

Results of this Research

As was discussed in the previous chapter, the core elements of the configuration manager are the state table and the rule table, and these data structures model the current administrative configuration and the desired administrative configuration respectively. These core elements of the configuration manager and the management interface provided by the SNMP are addressed in this chapter with respect to the questions enumerated in Chapter 1.

4.1 Modeling System State

Modeling the current and desired system state of the NES 800 was the key elements of the configuration manager. The desired administrative state, as modeled by the rule table, provided an event-driven version of the classic configuration file used by most ISs today. The current administrative configuration was modeled by the system state table and provided a synchronization point for the evaluation of the rule table. Finally, the physical hardware state was modeled by a group of objects

which allowed aspects of the IS, which were not modeled by the system state table, to be monitored.

4.1.1 Hardware State Model

The hardware state is determined by diagnostic subsystems external to the configuration manager and is modeled with rows in the configuration state table. While the state table is discussed more completely in the next section, it is useful to determine how effectively the physical, hardware state was modeled. In particular, the research question number 1, asked

“To what extent were the objects used to model the hardware state of the IS adequate for use in an event-driven, reactive configuration framework? ”

It is most useful to examine how effectively network interfaces were modeled in this configuration framework. Network interface configuration poses one of the most significant challenges to dynamic, reactive system configuration since they are a key element of an IS, and they embody much of the critical, “learned” or configurable, information about a network protocol used in a given network.

A simple example of this type of synchronized configuration is the setting of a network address on a given network interface. The address must be set after the hardware is initialized and before the associated network-layer protocol is initialized. This synchronization issue becomes more significant when multiple network-layer protocols are supported, and each has different data-link configuration require-

ments.

The state table proved to be a sufficient model of hardware state, exemplified mainly by network interface state, allowing synchronization of the various “interested” subsystems. The NES 8000 supports multiple network interface types and multiple network-layer protocols, in all cases the state table modeling of physical hardware state provided the necessary information required for synchronization.

4.1.2 Current Administrative State Model

Accurately modeling the current administrative state of an IS is complex, but it was found during this research that the system state table provided a useful, focused point for configuration synchronization.

As in the previous section, consider how configuration is performed for a network interface: when an interface becomes available, either due to system initialization or hot-swap, a message is sent to the configuration manager announcing the change. A hardware state object, discussed in the previous section, is modified and a well-known state object is updated. When the state object is updated an event occurs if a rule has been defined which depends on this transition, and the rule is evaluated allowing arbitrary management actions to be performed. In many cases, data-link subsystems must be configured before an interface is available to the network-layer protocols. The configuration state table provides a consistent method for each subsystem to inform other subsystems of internal state without requiring the subsystem to have knowledge of the layer “above” in the network stack.

Research question number 2 was:

“To what extent does the set of IS states monitored and controlled by the configuration manager accurately model the current administrative state and do these objects provide a sufficient basis for event-driven, reactive system management? ”

The subset of current administrative state modeled by the state table provided adequate information for configuration synchronization but did not completely model the current administrative configuration. While this may, at first glance, seem to be a deficiency, it proved to be a sound software design decision since the set of configuration data for a given subsystem was managed by the subsystem, thus isolating complexity.

The current administrative state, as modeled by the configuration state table, provided a sufficient basis for event-driven, reactive system configuration. In all configuration situations the state table was used to drive the reactive configuration manager.

4.1.3 Desired Administrative State Model

Modeling the desired administrative state in a reactive system is perhaps the most interesting aspect of this research. The framework provided by the configuration manager allows multiple configurations to be defined and applied as the current administrative state changes. For example, several configurations may be defined for a given network interface, and the “correct” configuration would be selected depending on the the physical hardware type associated with the indicated interface. Research question 3, asks

"How effectively is the desired administrative configuration modeled with the configuration rule table and to what extent does this framework allow arbitrarily complex system management actions? "

The desired administrative configuration was effectively modeled by the configuration rule table and proved to provide a sufficient mechanism for arbitrary management actions. The effectiveness of the rule table model of the desired administrative configuration can be judged by how consistently it was used to provide configuration data for the NES 8000. In all cases the desired administrative configuration data was stored in the system rule table. It should be noted that although some data, such as the manufacture date, was stored in non-volatile memory, independent of the configuration rule table, that data was not considered configuration information.

The configuration manager framework allowed management actions to be performed upon the occurrence of an event via two mechanisms: an SNMP set operation, or an event subroutine. As was discussed in Chapter 3, any valid SNMP set operation may be performed when an event occurs. This allows any of the well-defined management objects to be manipulated. In some cases it is more efficient to perform management actions directly, rather than performing an SNMP set operation. The event subroutine allows this by providing a way for subsystems to perform internal, management actions whenever a specified event occurs.

4.2 SNMP Suitability

The SNMP architecture is composed of a collection of one or more *network management stations*, *managed nodes (agents)*, and a network management *protocol*, which is used by the management stations and agents to communicate [Rose 91b]. Management stations control network elements such as hosts, gateways, and terminal servers which have SNMP agents responsible for performing the management functions requested by the management station [Case 93a]. This relationship was shown in Figure 2.1 in Chapter 2.

An important design requirement of the configuration manager was that a standard management framework be utilized. The SNMP was the natural choice for this framework since the Internet suite of protocols (TCP/IP) is the most widely used protocol suite [Dataquest 93] today, and the SNMP embodies the Internet standard network management framework. In light of this choice,

“To what extent is the Internet standard network management framework suitable for remote and local management of a reactive configuration manager. ”

A general limitation of network management is that the network must be functional, and the local configuration of network interfaces must be correct. Since the configuration of network interfaces was controlled by the configuration manager, and indirectly by network management, it was possible to mis-configure an interface and lose remote access to the IS. This was not considered a significant problem since it was expected that experienced administrators would be performing

network configuration and the NES 8000 provided some testing for configuration errors.

The SNMP Structure of Management Information (SMI) [Case 93b] specifies how the syntax and the semantics of data objects are defined utilizing the OSI presentation-level protocol ASN.1. By utilizing a subset of ASN.1, complex data types were avoided in the SNMP framework. The SNMP designers felt that encoding and decoding of many of the complex ASN.1 data types was not practical for many resource-limited systems. In light of this limitation it is useful to determine if configuration data could be represented with the SMI satisfactorily. In all cases the configuration data was easily mapped to SMI data representations.

4.3 Summary

This chapter has addressed the research questions outlined in Chapter 1. The first three questions addressed the key elements of configuration - physical hardware configuration, current administrative state, and desired administrative state. In addition to these issues the suitability of the SNMP was examined.

In the following chapter the results of this research are summarized presenting both positive and negative aspects of this work.

Chapter 5

Conclusions and Recommendations

In the previous chapter several key research questions were addressed, leaving more general conclusions and recommendations to be discussed in this chapter.

5.1 Conclusions

This research has suggested that reactive techniques may be used to simplify configuration of complex computing applications which require extensive, context-specific configuration data. Although some aspects of NES 8000 configuration framework, which are discussed in the following section, were found to be cumbersome, the framework did provide a suitable mechanism for event-driven, reactive configuration of a complex computing device.

The techniques utilized by the configuration manager to perform event-driven actions could be used in conjunction with fault-tolerant systems. By providing a

standard, configurable environment fault-tolerant systems could be more flexible, allowing administrators to configure fault-handling using standard network management tools.

Another potential use for the results of this research is the application of expert systems for monitoring and controlling networks. An internetworking IS would be the natural location for this function.

SNMP has proven to be a useful tool to system and network administrators in many instances. The SNMP-accessible framework developed with this research will build upon this success and allow administrators to utilize existing standard network management tools.

5.2 Recommendations

Some aspects of the configuration manager proved to be cumbersome, resulting in added complexity. Most notably, simple event grammars, the lack of temporal semantics for events, and the restricted subset of managed objects which could trigger events proved to be problematic and should be addressed in future work.

Events occurred only when the value of one of the well-known state objects transitioned to a trigger value. This proved to be a limitation since a more complete boolean syntax was not allowed, and other managed objects could not be used to trigger events.

Although any complex boolean expression can be represented by combining several simple boolean expressions, the resulting complexity is awkward for network

administrators. Moreover, the multi-step process required was less efficient than the evaluation of a single rule. Clearly, future reactive configuration framework should allow a more complete boolean event grammar.

In addition to the grammatical limitations of the event definition, the requirement that only well-known state objects could be used to trigger events was found to reduce the usefulness of the configuration system by reducing its flexibility for network administrators. This research suggests that all SNMP-manageable objects be allowed in the event specification clause.

Another issue that limited the flexibility of the configuration manager was the lack of temporal semantics for event definitions. It would be useful to express an event as condition **C** occurring within the last **S** seconds or perhaps within the next **M** minutes. This feature would add little complexity and allow rules to be defined for expected temporal events such as the failure of a command.

5.3 Summary

As was stated in Chapter Three, the requirements for the configuration manager were

efficient design - the configuration manager was implemented for an internet-working IS which had limited memory resources,

hot-swap support - some elements of the hardware configuration could be changed by system administrators without shutting down the IS,

SNMP manageable - the configuration framework must be completely accessible and manageable with the SNMP, and

allow arbitrary actions - complex actions must be performed upon the occurrence of events.

Each of these requirements was met by the implementation studied in this research. With the exception of the limitations listed in the previous section, this configuration framework proved to be flexible and suitable for the task.

Bibliography

Bibliography

- [Case 90] Case, J., Davin, J., Fedor, M., and Schoffstall, M., "Keeping it Simple," *Networking*, Volume 8, Number 3, March 1990.
- [Case 93a] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Introduction to version 2 of the Internetstandard Network Management Framework," Request for Comments 1441, April 1993.
- [Case 93b] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)," Request for Comments 1442 April 1993.
- [Case 93c] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Textual Conventions for version 2 of the the Simple Network Management Protocol (SNMPv2)," Request for Comments 1443, April 1993.
- [Case 93d] Case, J., McCloghrie, K., Rose, M., and Waldbusser, S., "Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)," Request for Comments 1448, April 1993.

- [Cisco 92] Cisco Systems, "Router Products Configuration and Reference," software release 9.1, Cisco Systems, Menlo Park, CA, September 1992.
- [Dataquest 93] Dataquest, "Survey of 301 Technology Managers," Dataquest publishing, April 1993.
- [Dulieu 87] Dulieu, M., Baily, M., Shaw, R., Leitch, R., "RESCU - A Real-Time Expert System," 7th International Workshop, Expert Systems & Their Applications, Avignon, France, May 1987.
- [Harel 85] Harel, D., Pnueli, A., "On the Development of Reactive Systems," Computer Science Department Technical Report, Carnegie-Mellon University, Pittsburgh, PA, January 1985.
- [ISO 87a] Information Processing Open Systems Interconnection Specification of Abstract Syntax Notation One (ASN.1). International Organization for Standardization, 1987. International Standard 8824.
- [ISO 87b] Information Processing Open Systems Interconnection Specification of Abstract Syntax Notation One (ASN.1) Draft Addendum 1: Extensions to ASN.1, International Organization for Standardization, 1987. Draft Addendum 8824/DAD 1.
- [ISO 87c] Information Processing Open Systems Interconnection Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). International Organization for Standardization, 1987. International Standard 8825.

- [Khanna 86] Khanna, R., Moore, R., "Expert Systems Involving Dynamic Data for Decisions," 2nd International Expert Systems Conference, Andover, MA, 1986.
- [McCloghrie 88] McCloghrie, K. and Rose, M., "Management Information Base for Network Management of TCP/IPbased Internets." Request for Comments 1213, March 1991.
- [Perkins 93] Perkins, D., "Standards," The Simple Times, Volume 2, Number 2, Dover Beach Consulting, Mountain View, CA, September/October 1993.
- [Postel 83] Postel, J., Reynolds, J., "Telnet Protocol Specification," Request for Comments 854, May 1983.
- [Postel 81b] Postel, J., "Internet Protocol," Request for Comments 791, September 1981.
- [Powell 91] Powell, D., "This May Be The Year Users Get What They Want," Networking Management, Volume 9, Number 1, January, 1991.
- [Proteon 92] Proteon, Inc., "Bridging Router Command Guide," Revision B, Proteon, Inc., Westborough, MA April 1992.
- [Rose 90] Rose, M., "Management Information Base Network Management of TCP/IP based internets: MIB-II," Requests for Comments 1158, May 1990.

- [Rose 91b] Rose, M., *The Simple Book: An Introduction to Management of TCP/IPbased Internets*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
- [Scott 90a] Scott, K., "Order to Chaos," *Data Communications*, Volume 19, Number 4, 21 March 1990.
- [Slattery 93] Slattery, T., "Friendlier than Ever," *The Packet*, Volume 5, Number 3, 3rd quarter 1993.
- [Taunton 86] Taunton, J., Haspel, D., "Rule Based Expert Systems - An Application in Real Time Process Control and Optimization," 2nd International Expert Systems Conference, Andover, MA, 1986.

Appendix

A configuration MIB for the NetEdge NES 8000

The configuration groups of the NetEdge MIB model the current administrative state with a state table and model the desired administrative state with a rule table. The configuration MIB module provides a standard way for tasks within the NES 8000 and remote management applications to monitor and control the configuration manager.

NES8000-CONFIG-MIB DEFINITIONS ::= BEGIN

IMPORTS

DisplayString FROM RFC1158-MIB
OBJECT-TYPE FROM RFC-1212
enterprises FROM RFC1155-SMI
config FROM NetEdge;

fcState OBJECT IDENTIFIER ::= { config 2 }
fcRule OBJECT IDENTIFIER ::= { config 3 }
fcModules OBJECT IDENTIFIER ::= { config 4 }

10

--

-- The system State table. This table provides a
-- way for EAS 8000 entities to inform the config-
-- uration manager, and in turn other EAS 8000
-- entities, their state. This table models the

-- *current administrative state.*

--

20

fcStateTable OBJECT-TYPE

SYNTAX SEQUENCE OF StateEntry

ACCESS not-accessible

STATUS mandatory

DESCRIPTION

"A table listing the state of various
EAS 8000 Entities."

::= { fcState 1 }

30

fcStateEntry OBJECT-TYPE

SYNTAX StateEntry

ACCESS not-accessible

STATUS mandatory

DESCRIPTION

"An entry describing the state of a
EAS 8000 Entity."

INDEX { fcStateIdent, fcStateInstance }

::= { fcStateTable 1 }

40

StateEntry ::= **SEQUENCE** {
fcStateIdent **INTEGER**,
fcStateInstance **INTEGER**,
fcStateValue **INTEGER**
}

fcStateIdent OBJECT-TYPE

SYNTAX INTEGER {

portState(1), -- *hardware states*

sysMemState(2),

rbMemState(3),

lbMemState(4),

50

smtState(100), -- *link-level subsystems*

spanningTreeState(101)

ipRouterState(200), -- *router subsystems*

ipxRouterState(201),

appleTalkRouterState(202),

snaRouterState(203),

60

```

decNetRouterState(204),

    -- add an enumeration for
    -- each monitored state.
}
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "A unique non-negative integer used
    to identify an entity in the EAS 8000
    system which can trigger an event."
 ::= { fcStateEntry 1 }
70

fcStateInstance OBJECT-TYPE
SYNTAX INTEGER (1..255)
ACCESS read-only
STATUS mandatory
DESCRIPTION
    "The instance of 'fcStateIdent' value.
    For example, portState(1) may have an
    instance of (1..40)."
 ::= { fcStateEntry 2 }
80

fcStateValue OBJECT-TYPE
SYNTAX INTEGER {
    -- Unknown state
    unknown-invalid(1),    unknown-testing(2),
    unknown-operational(3), unknown-configuring(4),
    unknown-failed(5),
90

    -- WAN / High-speed serial State
    hssi-invalid(10),    hssi-testing(12),
    hssi-operational(13), hssi-configuring(14),
    hssi-failed(15),

    -- Ethernet State
    enet-invalid(21),    enet-testing(22),
    enet-operational(23), enet-configuring(24),
    enet-failed(25),
100

    -- Token-ring State
    tr-invalid(31),    tr-testing(32),

```

tr-operational(33), tr-configuring(34),
tr-failed(35),

-- *T3 State*

t3-invalid(51), t3-testing(52),
t3-operational(53), t3-configuring(54),
t3-failed(55),

110

-- *FDDI port State*

fddi-invalid(61), fddi-testing(62),
fddi-operational(63), fddi-configuring(64),
fddi-failed(65),

-- *ATM port State*

atm-invalid(71), atm-testing(72),
atm-operational(73), atm-configuring(74),
atm-failed(75),

120

-- *General hardware State*

hw-invalid(101), hw-testing(102),
hw-operational(103), hw-configuring(104),
hw-failed(105),

-- *General subsystem State*

sys-invalid(201), sys-testing(202),
sys-operational(203), sys-configuring(204),
sys-failed(205)

}

130

ACCESS read-write

STATUS mandatory

DESCRIPTION

"The current state of this EAS 8000 entity.

This is an entity-specific value. For
example, when the state of a port
changes the value of 'fcPortHwStatus'
is used to set this object."

::= { fcStateEntry 3 }

140

--

-- *The EAS 8000 Rule table. This table models the*
-- *desired administrative configuration and*

-- reacts to changes in the state table.

--

fcRuleTable OBJECT-TYPE

SYNTAX SEQUENCE OF RuleEntry

ACCESS not-accessible

150

STATUS mandatory

DESCRIPTION

"A table listing rules which model the
configuration manager expert system."

::= { sysRule 1 }

fcRuleEntry OBJECT-TYPE

SYNTAX RuleEntry

ACCESS not-accessible

160

STATUS mandatory

DESCRIPTION

"An entry describing an expert system rule."

INDEX { fcRuleIdent, fcRuleInstance, fcRuleSequence }

::= { fcRuleTable 1 }

RuleEntry ::= SEQUENCE {

fcRuleIdent **INTEGER**,

fcRuleInstance **INTEGER**,

fcRuleSequence **INTEGER**,

fcRuleAction DisplayString,

170

fcRuleStatus **INTEGER**

}

fcRuleIdent OBJECT-TYPE

SYNTAX **INTEGER**

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The value of 'fcStateIdent' for which
his rule has meaning."

180

::= { fcRuleEntry 1 }

fcRuleInstance OBJECT-TYPE

SYNTAX **INTEGER** (1..256)

ACCESS read-only

STATUS mandatory

DESCRIPTION

"The value of 'fcStateInstance' for which this rule has meaning.

190

NOTE: The value (256) is reserved to mean ANY. For example a rule defined to trigger on portState(1).256 will be evaluated for any instance of the portState(1) identifier."

::= { fcRuleEntry 2 }

fcRuleSequence OBJECT-TYPE

SYNTAX INTEGER

200

ACCESS read-only

STATUS mandatory

DESCRIPTION

"This object defines the ordering of rules which pertain to a particular identifier/instance pair.

Instances of this object will always have the value (1..N) where N is the number of rules defined for a given identifier and instance pair. If a SET operation is used to create a rule for a sequence number greater than N it will be created at that location and then, immediately following the SET operation, be relocated to location N+1."

210

::= { fcRuleEntry 3 }

fcRuleAction OBJECT-TYPE

SYNTAX DisplayString

220

ACCESS read-write

STATUS mandatory

DESCRIPTION

"The rule to be executed/evaluated when an event is triggered. The rule is of the form:

'WHEN 4.5.6.0 == 12

SET 1.2.3.0 = 'hi mom', 1.2.4 = 12'

230

'WHEN 4.5.6.\$I == \$V

SET 1.2.3.\$I = "hi mom", 1.2.4 = \$V'

The wildcard values, \$I and \$V, may be used to indicate the state object instance and value in the SET clause if and only if they are first used in the WHEN clause."

::= { fcRuleEntry 4 }

240

fcRuleStatus OBJECT-TYPE

SYNTAX INTEGER {

valid(1),

invalid(2)

}

ACCESS read-write

STATUS mandatory

DESCRIPTION

"Setting this value to invalid(1) will delete the associated rule."

250

::= { fcRuleEntry 5 }

END

VITA

Michael Davison received his B.S. degree in Computer Science from Western Washington University and will be receiving his M.S. degree from the University of Tennessee, Knoxville. During his undergraduate work he participated in the implementation of a distributed, event-driven programming system on an Intel Hypercube, and worked as a systems programming intern with the Western Washington University Computing Center. His graduate work has focused on systems programming and aspects of system administration in a distributed, heterogeneous environment. He was a member of the Computer Science Department laboratory staff at the University of Tennessee, Knoxville and is currently employed by Net-Edge Incorporated.