Intelligent Agents for Network Fault Diagnosis and Testing

Garry Grimes^{*}

Euristix Ltd.

Century Court, 100 Upper St. Georges St., Dun Laoghaire, Co. Dublin

Tel: +353 1 288 4788 Fax: +353 1 288 5392 email: gary@euristix.ie

Brian P. Adley

Department of Electronic and Computer Engineering &

Telecommunications Research Centre

University of Limerick, Limerick, Ireland

Tel: +353 61 202609 Fax: +353 61 202572 email: Brian.Adley@ul.ie

Abstract

In modern complex networks remote control testing and diagnosis is necessary for efficiency. In this paper we propose the use of Intelligent Agents to carry out these tasks. These autonomous intelligent or programmable agents allow for full flexibility in unplanned conditions. They may be instructed to perform tasks at the resource thus reducing network loads in times of trouble and reducing the burden on the managing system.

The framework of existing network management standards is sufficient to implement such intelligent agents using only MIB descriptions based on existing standards. Tests may consist of one or several managed Intelligent testing agents and should be managed according to the situation. In this paper we look at distribution of testing scripts and control of the agents as well as the storing and returning of test results, in particular from distributed tests. We also present several practical implemented uses of the technology.

Keywords

Intelligent Agents, Test Objects, SNMP, Test Manager

1 INTRODUCTION

In order to accommodate unpredicted changes in the way networks are used and in their composition including both hardware and software components, flexibility is needed in the management system. In addition complex monitoring and control operations may be needed that involve multiple co-operating management objects on different platforms. Furthermore, in large networks a network management application may be topologically distant from the multiple objects that are performing a specific management operation. These requirements support the need for intelligent dynamic and autonomous agents to perform management operations. Intelligent Agent technology is being investigated in many distributed application situations and in this paper we investigate its application to Network Management. In particular, we have implemented and analysed the use of intelligent agents for fault analysis within current network management standards and architectures. This paper presents the results of this work and concludes that current management standards can support the use of Intelligent Agents.

2 CURRENT NETWORK MANAGEMENT APPROACHES

Standardised network management systems are implemented based on an object based manager-agent model,

^{*} This work was completed while at the University of Limerick

(Ashford, 1989) & (Gering, 1993). The manager requests or updates information on the state of a resource from an agent which represents that resource in the form of managed objects held in a managed information base, MIB.

The system is thus managed through a distributed database. The data is held in MIBs which are maintained by agents. The agents running on the nodes are in effect database interfaces. They update their own MIBs depending on what is happening with the real resource that the MIB represents and they allow the managing system to query this information. The agents cannot make decisions or initiate any management activities, this is left completely to the managing system. Where information is to be gathered from the MIBs of several agents and compiled for analysis it is the managing system which will perform all the operations involved; querying the agents, receiving the results and compiling them.

3 MANAGING TESTING AND DIAGNOSIS FUNCTIONS

3.1 Problems with the established architecture

In most cases only a single agent per resource will be required to execute the test function selected by a manager, for example (Noto, 1996). In the case of end-to-end path testing however, tests may be required that necessitate the co-operative participation of several distributed management agents. This can lead to the following problems (Vassila, 1995) & (Koojiman, 1995):

- Many management operations are necessary in a complex test situation. With the manager performing so many management operations on several geographically distant agents, network bandwidth usage is concentrated at the management system. Also testing functions on several agents may produce large quantities of resulting data to return to a manager. This is not good, especially where such tests are being carried out in serious failure situations.
- Where all control is centralised to one managing station this station (and the link to it) becomes of critical importance where the carrying out of a test must be orchestrated by the manager which initiated it. This would be the case where agents simply store and convey information. Should the manager or its link experience a failure, all operations by the agents, including any tests-in-progress must terminate or suspend until normal operation resumes.
- Manager processing power is limited. Greater loads on managing systems, such as from processing test result data from several agents means less available for other management tasks.

The processing power and operational resources available to agents of modern managed devices are reaching the stage where they are capable of far more complex activities than ever before. The SNMP RMON MIB (Waldbusser, 1991) allows for this to a certain degree. This standardised MIB allows for fault management to the extent that a manager may request data on network interface traffic from managed devices and summarisations from the agent which has been monitoring the network traffic. This information is very useful for fault management, diagnosis, and testing, but only standard information is available. Should more complex or specific testing be required this approach lacks the flexibility to adapt to requirements.

3.2 Improvements to the established model: The Intelligent Agent

Much work has been done, including (Vassila, 1995), (Magedanz, 1995), and (Meyer, 1995), into solutions to the problems of centralised network management and improvements in flexibility to extend the functionality of current managed systems. Such approach is shown in figure 1. The central idea here is the delegation of tasks which would normally be explicitly controlled by a management station, to the managed agents. This is achieved by incorporating a facility into the agents which allows them to download scripts of instructions and to interpret and execute them. With this architecture agents become capable of performing tasks autonomously with or without the co-operation of other managed systems. Should managers require several agents on a remote system or network to work together to perform a task, the managing system can delegate responsibility for control of the task to one of the managed systems. The manager thus instructs the agents involved in the tasks they are to perform. It should also be possible to indicate a certain course of action to be taken in the case of exceptional circumstances arising, such as failure of a system involved in a test.

Figure 1 Architecture of an Intelligent Agent.

With such a scenario the performance of the managed network would be increased as the gathering and processing of data is moved nearer to the source of that data. Full flexibility is achieved where agents can be instructed 'on-the-fly' to carry out specialised tasks. Also many of the weaknesses in the established model of use are resolved:

- Where sets of management instructions are performed autonomously by an agent, management functionality is delegated away from the manager. The only management information flowing between the managing system and the agents is the initial set of management instructions and the final results of the management operations which have been carried out by an agent or agents. Network traffic concentration at the managing station is therefore greatly reduced.
- Once the management instructions have been communicated to the agent or agents involved, then faults in the management station or in the link between it and the agents will not be catastrophic as the agent processes can continue to function independently. When the operation results have been compiled they will be available to the manager as soon as the situation has been resolved and proper functioning has resumed.
- The processing load on the manager is greatly reduced where the bulk of the data gathering and manipulation is done by the agents rather than the manager. A managing system may therefore initiate many such operations at no extra overhead and wait to be informed by an agent when it has completed all the necessary operations and the final results are available. It may also perform intermediate checking of results to ensure that all is proceeding as it should.

4.0 TESTING BY INTELLIGENT AGENT

4.1 Manager-Agent relations in testing

Using an Intelligent Agent system for testing and diagnosis will involve different manager-agent relations depending on the type and extent of the tasks to be carried out. Different manager-agent relations for a distributed test have been discussed by Ohta et al. in (Ohta, 1993). In the paper the authors state that the optimum form of management will depend completely on the application scenario. Greater system processing loads favouring delegation of tasks away from the manager.

With a test involving distributed co-operating agents we found that delegation of manager status to one participating (or non-participating) agent works best. Using a test-manager agent to co-ordinate a test and retrieve and process results cuts down on processing load of and network traffic to the main manager system. Test control in the event of necessary intervention by the main manager is simplified. Should the main manager have to suspend or terminate a test then control of all testing agents is effected by communication with the delegated test-manager agent. The diagrams in Figure 2 and Figure 3 below show the manager-agent relationship as direct, Figure 3 and with a test-manager in Figure 2. A test using the architecture of Figure 3 with n testing agents will require n sets of management communications for each command to begin, suspend, resume or terminate the test. The manager must also have a understanding of the functions of each testing agent and know in which order commands to agents should be sent. For example, in the tests described in Section 8.1 the agent producing test traffic should be suspended before the traffic analysis agent.

Figure 2 Manager-Agent relations with a delegated test manager agent

Figure 3 Manager-Agent relations with no test manager agent

In the case when a test can be done using one managed agent, such as with the Broadcast Storm source locator function (see Section 8.3) it was found that a direct manager to agent relationship works best. When using an Intelligent Agent for testing there is nothing to be gained by delegating a further test-manager agent expressly for controlling one testing agent or retrieving and processing results. Such functions can be incorporated into the testing agent script itself.

4.2 Structure of a test-manager

Each intelligent agent has the necessary functionality to take on the role of test manager. In our particular implementation delegation was arbitrary, however in practice there will be external conditions which affect the decision as to which particular agent should be given this responsibility. There are many issues to consider; including base processing power, and average load on each agent's processor, as well as the average traffic through its network interfaces. Such discussion is outside the scope of this paper but should be considered for future work.

The test manager agent has various tasks to perform regarding test set up and control, as well as its own testing operations. These various steps which must be taken by the test manager could be coded as part of the agent itself. This however makes testing less flexible with the setting up, running and completion of tests being of a set procedure, and reporting of results having the same format. In order to introduce enough generality into this method of control it would be necessary to provide large quantities of test related information by setting MIB objects. Many tests will require specific information for set up and dealing with exceptional circumstances. Statically defining and creating specific MIB objects to store all this information is not desirable. Most of the more specific variables will remain unused in many circumstances and it is not realistic to assume that all test conditions and criteria can be anticipated and catered for in this way. Having said this, it is useful to provide some general 'lowest-common-denominator' test information as MIB objects. General object descriptions for test functions have been detailed in (ISO, 1993) & (NMF, 1992).

In contrast to a fully MIB specified approach to the architecture of a test manager, a more flexible and straightforward method is to develop a script for any agent which will deal with the set up and control of tests. Supplying this script to an agent which will execute it effectively makes that agent into a test-manager. In order to do this a special control script is sent to the delegated test-manager. This instructs the test manager in how the test is to be set up among the agents involved, how unusual circumstances are to be dealt with and when the test should be initiated and completed. It should also instruct how to obtain and process results from each agent which has been involved in the test and can notify the main manager when final results are available.

5.0 IMPLEMENTATION

5.1 Why use a Standardised Network Management Protocol ?

There are many benefits to be gained from standards-based integrated network management; more rapid deployment of new services and integration of these services into existing systems and a common management platform across heterogeneous environments (Shrewsbury, 1993).

As Intelligent Agent Testing and Diagnosis is a service which may span diverse systems we were interested to see if an Intelligent Agent System could be incorporated into an existing network management framework and to this end decided to base an implementation on an existing management protocol.

After considering SNMP, the Simple Network Management Protocol (Rose, 1990), (Case, 1991) and CMIP, Common Management Information Protocol (ISO, 1991) network management platforms we decided to use SNMP. The CMIP standard would have allowed for a very capable and well defined system, SNMP however supported our requirements with less overhead, allowing more time to investigate the issues involved.

5.2 The Scripting language

Because of its network management extensions and its widespread use and support the Intelligent Agent System was implemented using Scotty. This is the Tool Command Language developed by John Ousterhout (Ousterhout, 1990) with extensions developed by Juergen Schoenwaelder (Schoenwaelder, 1995).

The security and integrity of a network node becomes an issue when it provides the capability to execute scripts remotely. In the Intelligent Agent System this is somewhat resolved by not allowing 'dangerous' commands to directly access the underlying operating system of a device. Such commands are either not allowed or put into 'wrappers' which allow only certain 'minimum-damage' functionality. Also scripts are allowed to write to and read from files in only one local directory, this works in a way similar to the HTTP access of World Wide Web home sites.

6 DESIGN OF THE MIB

The requirement was to develop a MIB which would allow the downloading of scripts to the agent, the control of this script and the returning of results when the script was complete.

6.1 Script downloading

There were several options available as regards script downloading. Among these HTTP, FTP, TFTP, SMTP and SNMP itself.

The function of distributing scripts to intelligent agents requires a mechanism for transporting text files between the managing and managed systems. Several efficient TCP/IP protocols have been developed which can serve this purpose; HTTP (Berners-Lee, 1996), FTP (Postel, 1985), TFTP (Scollins, 1992), SMTP (Postel, 1982) and SNMP itself.

We wished to implement a script transfer mechanism using only SNMP to investigate whether the protocol would be sufficient for such a purpose, thereby removing the requirement for supporting additional ones. For the purposes of comparison we also decided to implement script downloading using a protocol developed specifically for ASCII text transfer. We selected HTTP for this purpose as it is supported by Scotty and is based on a simple model; set up connection, transfer data, close connection, which exactly fits the requirements of script transfer.

6.2 Script transport and the MIB

Using SNMP for script downloading requires breaking up scripts into lines which will then be performed by executing a specific line or by concatenating several lines together if they belong to the same script. This is necessary because of the message size limitation in the SNMP PDU. Scripts are broken into consecutive lines and are transported to the agent by the manager in separate Set commands. These scripts are stored in the MIB Table. Separate scripts are stored as conceptual tables within the main table. When a script is completely downloaded it is available for execution. To facilitate re-use the agent may be instructed to transfer a script to some non-volatile storage. To this end, the MIB contains an action object which on changing value (via a SET) instigates the I/O operation. To allow later retrieval of stored scripts the agent creates and stores a permanent script handle.

Transport of scripts with HTTP is more straight-forward and requires only two MIB variables as it is not stored in the MIB at all for transfer (though this could use a process similar to the SNMP download but from non-volatile storage). One variable will be SET to indicate the uniform resource locator (URL) of the location of the script to be downloaded, the other will hold any error messages that may occur during transfer. The managing system which supplies the script must have a HTTP server process running which is capable of responding to requests by the agent. When the script URL variable is set the agent issues a HTTP get command and stores the

script on non volatile storage. In our implementation which is file system based, this involved extracting the local filename from the URL as a handle for storage and retrieval. Depending on the existence of non-volatile storage and the type available, this operation should be performed appropriately by the agent in a manner transparent to the managing system.

In this investigation into methods of script transport we found that both SNMP and HTTP have the functionality to deal with the downloading of scripts to agents. It is possible to implement an intelligent agent system without recourse to a protocol other than SNMP. However, attempting to use a management protocol like SNMP to perform a task for which FTP, TFTP, and HTTP were developed is subject to several inefficiencies. The drawback lies with the increased overhead on the agent process. It must create and fill MIB table rows with each script line. These lines must then be extracted from the MIB and concatenated together to form the original script. The use of a more suitable protocol for transport such as HTTP, allows direct receiving and storing of a downloaded script without the use of the MIB as an intermediary. The extra process of table row creation, setting, extraction and concatenation is not required. If it is possible to include support in an intelligent agent for script retrieval via HTTP, or another such lightweight file transport protocol, we believe this to be far more efficient than the use of SNMP for the same purpose.

6.3 Control of the Script

As indicated by the specifications in (NMF, 1992) for a test management service, tests may be both implicitly or explicitly controlled. Implicit control may be script based, where an agent is instructed to monitor conditions in its controlling script. It may issue commands to other agents which are co-operating with it in the running of the test using explicit control commands. Also provided is the facility to specify when a script should begin to iterate and when to end, intervals after which it should iterate and number of iterations. Explicit control is provided by a SET on the action variables which control script execution.

6.3.1 Scripts are pre-prepared for execution

To use a downloaded script it is necessary to provide information for its use. A table is maintained with all this information, and the same script may be used in different ways depending on this information. The MIB has a scriptTable for this purpose. Each row of this table holds as well as the script name; its status, owner, number of iterations of execution and interval between each, and also input (arguments) to the script. As scripts need to be reusable under different conditions they are written to accept arguments. These are most likely to specify the names or addresses of other network devices when scripts are for path testing purposes and will be different from host to host.

6.3.2 Execution monitoring and control

Described in (ISO, 1993) is a model for managing the invocation of tests on remote resources. These tests may be controlled and subject to monitoring, suspension and resumption during application or uncontrolled where results are returned on test completion. This remote control test specification is intended for an OSI system and the corresponding CMISE network management system with its more complex GDMO MIB description format. The Network Management Forum produced OMNIPoint1 which provides a similar specification for a Testing Management Function. This specification though with ISO standards in mind is not explicitly intended to be implemented in GDMO.

These models have not addressed distributed co-operating agents within tests but do suggest a framework for any generic test that should be required to be performed. Because of this genericity they were particularly applicable to the use of testing by intelligent agents for which testing is not restricted to any one function or set of functions. Though we developed the intelligent agent to use SNMP as its management protocol with its MIB written using ASN1 the ITU-T and NMF test management models where not GDMO specific are still applicable to an SNMP MIB. In particular X.745 and OMNIPoint2 specify the testing procedure as requiring several states: *Not Initialised, Idle, Initialising, Suspended, Testing, Terminating and Disabled*. Testing actions are also specified in X.745 as *Initiate a test, Suspend, Resume or Terminate a test*.

When a script in the scriptTable is to be executed its ID is SET to the current script id variable. Control of the current script is by the RMON method of setting 'action' variables.

To control execution the currentScriptAction variable may be set to one of five values:

- Static - The initial value, also if there is no script or the script has not yet been run.
- Running - The current script is to execute.
- Suspended - The current script is to be Suspended before the next iteration.
- Resumed - The suspended current script is to resume operation where it was suspended.

- Terminated - The current script is to cease execution after the current iteration.

The script condition variable shows the current state of the script. This will indicate whether the script is: *static, executing, about to suspend, suspended, resumed from suspension, about to terminate or terminated.*

6.4 Return of Results

Intelligent agents which perform management tasks that are delegated to them by a managing system often do not require that results be returned. Research work thus far on the use of intelligent agents, including their use in performing network management tasks has not addressed this. Standardised specifications which address testing issues assume well defined procedures and well defined results.

When considering testing using downloadable scripts it is essential that results be available to management, even where a testing system is quite autonomous it is always necessary that a management station be able to monitor results.

Where tests are performed using downloaded scripts where there can be no precise standardisation, there can be no pre-arranged structure for storing or reporting results. Every script is different and performs a different function and will produce different results.

Two solutions exist within our framework for result reporting of scripted tests:

1. Output results to a file and export via HTTP to the manager.
2. Place results in a MIB and allow them to be retrieved via SNMP.

It is a simple matter to write out results to a file as they are obtained and indicate the result filename or URL for retrieval by the manager. The results file may then be displayed or parsed by a management application.

To make results available via SNMP it is necessary to store them in a MIB. However MIBs are developed for a particular purpose. Though it is possible to develop a MIB 'on the fly' specially prepared for storing script results and download it along with the script itself, this is not a realistic solution. SNMP agents are not intended to load MIBs at any time other than on initialisation. If a new MIB is loaded the agent must be reset and all previously obtained information will be lost. The Intelligent Agent System therefore uses a configurable result table in which script results may be stored, regardless of format. The table rows are created to store what ever type or format of result is required and can be retrieved via SNMP GETs or SNMPv2 GET-BULK.

6.5 Presenting Test Results to a User.

A problem with retrieving data from MIBs generally is that unless the retriever knows the MIB structure and the purpose of each variable and their relationships, it is difficult to transform the data into easily understandable information. This problem is compounded by the returning of results in an unknown format by an intelligent agent.

To allow for more 'user-friendly' reporting of results the Intelligent Agent System allows for the downloading of a Result-Display-Script by the (test) script developer to the individual agents or to a controlling agent which will gather final results.

This script is available to a managing system for uploading when the script has results available to specify exactly how result data should be presented to a human user. These Display Scripts are exported to the agent in the same manner as the Scripts that the agent runs. Uploading of them has been provided for by HTTP and by loading into a Display Script Table maintained by the Agent and from which the Display Script may be retrieved using SNMP GETs.

The IETF has taken into consideration the uploading of user interfaces written in Java (Gosling,1995) for management by Web browsers, and indeed our display scripts may be written in any interpreted language which provides an SNMP interface. However, the Display Scripts developed during this project use the Tcl extensions Tk with the extra Scotty functionality.

7.0 THE INTELLIGENT AGENT PROGRAM

The agent program uses the network management extended Tcl interpreted language Scotty. Scotty is based on event driven programming. Typically a Scotty application will load an initialisation script which installs some event handlers. In the case of an SNMP agent, these event handlers are configured to respond to SNMP PDUs received and interface with the MIB database to update or return MIB variables. Once initialisation is complete

the application enters an event loop. The event driven approach works well in most cases however one problem is the lack of threads. New processes must be explicitly started to deal with SNMP requests otherwise the listening process will block until the request has been completed. In the Intelligent Agent System Event handlers deal with making available table contents, retrieving and setting MIB variables and performing explicit script control actions.

The Scotty extensions allow for very comprehensive network testing and include interfaces to TCP/IP services like UDP Datagrams, TCP message passing, ICMP services as well as HTTP and SNMP. All these are available to downloaded scripts.

7.0 MANAGEMENT STATION APPLICATION

The management can be performed by using the Scotty command line interface for control and this was adequate during development. However the Tk toolkit that forms a part of the Tcl suite of extensions makes all such commands available for control by GUI. A manager application was developed which really provided a graphical front end to the Scotty SNMP GETs and SETs but specifically developed for the control and monitoring of the Intelligent Agent by its MIB.

8.0 TESTS IN PRACTISE - DESIGN OF TEST SCRIPTS

Four applications of the Intelligent Agent System to Network Fault Diagnosis and Testing were developed to better investigate the use of fault and performance testing with an intelligent agent system. These functions were: A distributed one way UDP through-put test, a distributed loss rate test, a broadcast storm locator, and resource utilisation over time.

8.1 Distributed one way through-put test (UDP)

The purpose of this test is to determine the maximum UDP datagram throughput of a link. It is a distributed function using two testing agents; the datagram traffic generator at one end of the link and the datagram receiver at the other end. In addition there is a test manager agent which runs the distributed function at regular intervals until the required results have been obtained. These results are then retrieved by the test manager agent and summarised (average throughput over time), the main manager is notified and can retrieve the final results.

It works as follows: A script is downloaded to an agent at the test traffic source to send as many datagrams as possible within a certain time period (n seconds). The agent at the test traffic destination point is instructed to receive as many datagrams as possible from the time of receipt of the first datagram until the test time (n seconds) has elapsed. A test manager agent is delegated which sets off these tests at regular intervals, a main manager request to suspend, resume or terminate the test is directed only to the controlling test manager agent which also gathers the end results from the datagram source and destination agents. It then calculates the throughput from *datagrams received / time*.

This method works well for in-service testing as test traffic has to travel only one way to be examined, thus reducing test traffic to half of loopback. Importantly, this also allows true one-way throughput examination rather than duplex. Explicit manager control of the test and retrieval of results is by communication with the delegated test-manager. Test scripts for datagram traffic generation and receiving are shown in figures 4 and 5.

Figure 3 Architecture of the distributed UDP throughput test

8.2 Distributed loss rate test

This was mainly developed with ATM testing in mind and was inspired by the Ohta et al paper though it was (by necessity) implemented over TCP/IP using UDP datagrams. The test involved setting up a test traffic insertion point, a drop and analysis point and several intermediate monitoring points. All of these functions are distributed amongst agents along the path to be tested and are controlled by a Test-Manager-Agent which starts the test at various intervals and which collects results from all of the (sub) test-agents. This Test-Manager-Agent is so due to its own controlling script giving it that function. Explicit control of all agents (i.e. the whole distributed function) is effected by control of the Test-Manager-Agent. This test is distributed so that one way testing is possible, this is necessary in assymetrical point-to-point ATM links where regular loop back tests are not possible.

8.3 Broadcast Storm source location

Monitors network traffic, at packet level and logs all broadcast traffic on a sliding window basis over the last n packets. Should broadcast packets reach unacceptably high (configurable) levels it archives the last m packets (since just before the storm began) and makes them available for examination by a managing station. It also examines the contents of the archived logs to determine the source of the initial broadcast packet of the storm. This however only works in the situation where the broadcast storm is of RESENDS (which have identical data contents).

As a broadcast storm will often cut off communications the agent is completely autonomous and will continue monitoring, logging and processing relevant data until communication with the manager has been restored and the results can be retrieved.

8.4 Simple resource utilisation over time

An agent on each host monitors CPU utilisation though this may be easily extended to include monitoring of any quantifiable device value. There is no interfering packet transfer while the monitoring goes on, information may be obtained from the management station whenever required. This was really developed to test the uploadable display script functionality. All results from each CPU testing agent can be presented together in graph format from the results compiled by the central controlling agent which holds the display script.

9.0 RESULTS AND CONCLUSIONS

In this paper we investigated the application of Intelligent Agent technology to remote network testing and

diagnosis functions. We have found that the use of intelligent agents provides complete flexibility to develop and implement precise testing functions to deal with unforeseen situations and requirements. In distributed testing where several agents are involved in a test it was found to be very useful to delegate a test-manager agent to control testing and retrieve results. This reduces test management traffic (control and result data), main manager processing load, and also allows for an autonomous testing system.

We also found that it is not useful to develop a specific agent to deal with control of intelligent agents for testing. We find that fully configurable tests require fully configurable management, and that a scripted test manager is the best approach.

Configurable tests require configurable result storage which cannot be planned for in advance. We used a result table with agent configurable rows to solve this. The main manager application must be able to interpret these results if they are to be of use. We investigated a method of including a test specific 'display script' with the script of the delegated test manager agent. This can be uploaded by the manager and presents results in a human-readable format with text and graphics.

In this investigation we were interested to see if a complete Intelligent Agent system to meet our requirements could be developed within the framework of an existing network management architecture such as SNMP. Such a system is not intended to replace the static agent architecture but to work with it. Either as an extensible agent or simply listening for SNMP communication on a different UDP port as was implemented here.

We found that it is possible to implement such a system within the SNMP framework with no changes to the protocol. A MIB was developed according to the SNMP specifications (Stallings, 1993) with objects to hold all the required test information (including test script) and objects to control test functioning. Though script distribution was found to be possible using only SNMP it is more reliable and efficient to use HTTP. This was provided for in our implementation.

Due to our requirement to base the Intelligent Agent system on an existing network management standard we chose SNMP over CMIP mainly because of the widespread use of SNMP and the complicated and heavy-weight nature of CMIP. Scotty proved an excellent development platform for the system and provided excellent support for SNMP, HTTP, UDP and TCP. However an SNMP intelligent agent based platform is rather strained and has its drawbacks. The nature of actions (e.g. script control) not being an integral part of SNMP, they must be implemented by changing MIB object values. Action parameters in SNMP must be implemented by MIB objects which are explicitly SET before actions can be performed. Retrieval of large amounts of data is less efficient than CMIP and 'round-about' methods must be used for data reduction. The presentation of result data in table object format also seems rather tortured when compared with the GDMO Test results record object class described in X.745 (ISO, 1993).

In the course of this work we used Tcl to create our test scripts. This proved adequate for the purposes of testing but it would be useful to investigate other interpreted languages such as *telescript*, Perl and Java. Our test scripts were relatively small and involved minimum complexity, they were quite applicable to the simple unstructured form of Tcl. Large scripts are very unwieldy when written in Tcl and become very difficult to develop, maintain, and debug. The safety features of Tcl (latest version) with respect to agent scripts are very useful, however other languages, such as Java appears to address this area better.

10 ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of Ericsson System Expertise Ltd. and Teltec Ireland without whom this work would not have been possible.

11 REFERENCES

- Ashford, C. (1993) The OSI Managed Object Model. *ECCOOP '93*.
- Berners-Lee, T. Fielding, R. Frystyk, H. (1996) Hyper Text Transfer Protocol, Internet Working Draft, February 1996.
- Case, J. Fedor, M. Schoffstall, M. Davin, J. (1991) RFC1213: A Simple Network Management Protocol.
- Gering, M. (1993) CMIP vs SNMP. *Integrated Network Management III*, 347-359.
- Gosling, J. McGilton, H. (1995) The Java Language Environment: A White Paper, Sun Microsystems.
- ISO/IEC DIS10164-12 (1993) Test Management Function.
- ISO/IEC9596 Information Technology (1991) Recommendation X.711 - Open Systems Interconnections - Common Management Information Protocol Specification.

- Koojiman, R. (1995) R. Divide and Conquer in network management using event-driven network area agents, Technical University of Delft, The Netherlands.
- Magedanz, T. (1995) On the Impacts of Intelligent Agent Concepts on Future Telecommunications Environments, *IS&N '95 Conference Proceedings*, 396 - 414.
- Meyer, K. Erlinger, M. Betsier, J. Sunshine, C. Goldszmidt, G. Yemini, Y. (1995) Decentralizing Control and Intelligence in Network Management, *4th International Symposium on Integrated Network Management*, Santa Barbara, CA.
- NMF (Network Management Forum): Forum 012 (1992) Application Services: Testing Management Function, Issue 2.0.
- Noto, M. and Tesink, K. (1996) Definitions of Tests for ATM management, *IETF Internet Draft*.
- Ohta, S. and Fujii, N. (1993) Applying OSI systems management standards to virtual path testing in ATM networks. *Integrated Network Management III*, 629-640.
- Ousterhout, J.K. (1990) Tcl: An Embeddable Command Language, *USENIX '90 Conference Proceedings*.
- Postel, J. Reynolds, J. (1985) RFC959: File Transfer Protocol.
- Postel, J.B. (1982) RFC821: Simple Mail Transfer Protocol.
- Rose, M. McCloghrie, K. (1990) RFC1155: Structure and Identification of Management Information for TCP/IP-based Internets.
- Scollins, K. (1992) RFC1350: Trivial File Transfer Protocol.
- Schoenwaelder, J. and Langendorfer, H. (1995) Tcl Extensions for Network Management Applications, Technical University of Braunschweig, Germany.
- Shrewsbury, J.K. (1993) TMN in a nutshell.
- Stallings, W. (1993) SNMP, SNMPv2 and CMIP: The practical guide to Network Management Standards, Addison-Wesley.
- Vassila, A. and Knight, G.J. (1995) Introducing Active Managed Objects for Effective and Autonomous Distributed Management, in *IS&N '95 Conference Proceedings*, 415 - 430.
- Waldbusser, S. (1991) RFC1271: Remote Network Monitoring Management Information Base.

11 BIOGRAPHY

Garry Grimes obtained the BSc in Computer Systems from the University of Limerick in 1994. He was awarded the MEng by research and thesis in telecommunications in 1996 from the University of Limerick. This was awarded on the basis of research into the application of Intelligent Agent mechanisms to standardised network management architectures. This work was mainly directed at building on current fault and performance management techniques in TCP/IP managed networks. Garry is now based at Euristix Ltd., Dublin, Ireland, where he is involved in the design and development of enterprise network and element management solutions.

Brian Adley has been member of the Department of Electronic and Computer Engineering faculty at the University of Limerick from 1992. Between 1975 and 1992 he has held a number of communications related positions in engineering and management including Hewlett Packard Inc., Apollo Computer Corp, Protein Databases Inc., ICL and Marconi Space and Defence. He holds a B. Sc. (Hons) in Computer Science from De Montfort University (1975) and a M. Sc. Computer Science from Boston University (1991). His current interests are distributed management and image processing.

```

##
## Send a result back to IAgent
##
proc setresults { str1 res1 } {
    global rf
    udp send $rf { [ list str1 res1 ] }
}

proc udp_send { file host delay } {
    global msg stat size
    if {[catch {udp send $file $msg} err]} {
        catch {
            udp close $file
        }
        return
    }
    incr stat(send,$host) $size
    if {$delay > 0} {
        after $delay "udp_send $file $host $delay"
    } else {
        after idle "udp_send $file $host $delay"
    }
}

proc udp_summary { file host secs } {
    global stat sdt
    udp close $file
    set send $stat(send,$host)
    set o_speed [expr $send * 8.0 / 1024 / 1024 / $secs]
    puts [format "%6.3f MBit/s send" $o_speed $host]

    set info "udpsend output results line"
    ## Create the output summary line
    set sn "SN:udpsend.tcl"
    set fdt "FDT:"
    append fdt [exec date]
    set ds "DS:"
    append ds $send
    set dur "DUR:"
    append dur $secs
    set spd "SPD:"
    append spd $o_speed

    set summ [list $sn $sdt $fdt $ds $dur $spd]
    puts $summ
    ## Update the resultTable with output
    setresults $info $summ
}

##
## Connect a udp file handle to the test port on host and
## send datagrams
## to it. Calculate the send rate in kB per second.
##
proc udp_test { host secs len delay } {
    global msg stat size testport
    set size $len
    set testport 5556
    set msg ""

    for {set len 0} {$len < $size} {incr len} {append msg "+"}
    set stat(send,$host) 0
    if {[catch {udp connect $host $testport} f]} {
        puts stderr "host: $f"
        return
    }
    after [expr $secs * 1000] "udp_summary $f $host $secs"
    udp_send $f $host $delay
}

##
## Parse the command line arguments and call udp_test
## for every host given on the command line.
##
global sdt
set secs 10
set delay 10
set bytes 1024
set newargv ""
set parsing_options 1
while {(llength $argv) > 0} && $parsing_options {
    set arg [lindex $argv 0]
    set argv [lrange $argv 1 end]
    if {[string index $arg 0] == "-"} {
        switch -- $arg {
            "-d" { set delay [lindex $argv 0]
                    set argv [lrange $argv 1 end]
            }
            "-t" { set secs [lindex $argv 0]
                    set argv [lrange $argv 1 end]
            }
            "-s" { set bytes [lindex $argv 0]
                    set argv [lrange $argv 1 end]
            }
            "--" { set parsing_options 0
            }
        }
    } else {
        set parsing_options 0
        lappend newargv $arg
    }
}
set argv [concat $newargv $argv]
set sdt "SDT:"
append sdt [exec date]
if {$argv == ""} {
    puts stderr (usage: udpspeed [-d delay] [-t seconds] [-s size] hosts)
} else {
    set time 1
    foreach host $argv {
        after [expr {$time * 1000}] "udp_test $host $secs $bytes $delay"
        incr time $secs
        incr time 1
    }
}

```

Figure 4 Script to send datagram traffic.

```

##
## Send a result back to lAgent
##
proc setresults { str1 res1 } {
    global rf
    udp send $rf { [ list str1 res1 ] }
}

proc udp_receive { file host } {
    global stat size
    udp receive $file
    incr stat(received) $size
}

proc udp_summary { file host secs } {
    global stat sdt
    fileevent $file readable ""
    udp close $file
    set received $stat(received)
    set i_speed [expr $received * 8.0 / 1024 / 1024 / $secs]
    puts [format "%6.3f MBit/s received " $i_speed $host ]

## Prepare results summary info
set info "connection analysis output iteration summary"
set sn "udprecv.tcl"
set fdt "FDT:"
append fdt [exec date]
set ds "DS:"
append ds 0
set dr "DR:"
append dr $received
set dur "DUR:"
append dur $secs
set spd "SPD:"
append spd $i_speed
set summ [list $sn $sdt $fdt $ds $dr $dur $spd]
puts $summ
## Update the resultTable with output
setresults $info $summ
udp close $f
exit
}

##
## On receipt of the first datagram, start the timer.
##
proc firstreceipt { f host } {
    global secs
    fileevent $f readable "udp_receive $f $host"
    after [expr $secs * 1000] "udp_summary $f $host
$secs"
}

##
## Connect a udp file handle to the test port on host and

## receive datagrams
## from it.
##
proc udp_test { secs len } {
    global msg stat size testport
    set size $len
    set host [exec hostname]
    set testport 5556
    set stat(received) 0
    if {[catch {udp open $testport} f]} {
        puts stderr "$host: $f"
        return
    }
    fileevent $f readable "firstreceipt $f $host"
}

##
## Parse the command line arguments and call udp_test
## for every host given on the command line.
##
global sdt rf
set secs 10
set bytes 1024
set newargv ""
set parsing_options 1
while {[length $argv] > 0} && $parsing_options {
    set argv [lindex $argv 0]
    set argv [lrange $argv 1 end]
    if {[string index $argv 0] == "-"} {
        switch -- $argv {
            "-t" { set secs [lindex $argv 0]
                    set argv [lrange $argv 1 end]
                }
            "-s" { set bytes [lindex $argv 0]
                    set argv [lrange $argv 1 end]
                }
            "--" { set parsing_options 0
                }
        }
    } else {
        set parsing_options 0
        lappend newargv $argv
    }
}
set argv [concat $newargv $argv]

set sdt "SDT:"
append sdt [exec date]

## Create a UDP file handle for the result
set rf [udp open 5000]

## Begin the test
udp_test $secs $bytes

```

Figure 5 Script to receive datagram traffic.