DEBUGGING SCRIPTING APPLICATIONS FOR BATTLEFIELD NETWORK MANAGEMENT

Adarshpal S. Sethi
Dong Zhu

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

I. Introduction

Our research over the past three years of the ATIRP Consortium has resulted in the design and implementation of a hierarchical network management system called SHAMAN (Spreadsheet-based Hierarchical Architecture for MANagement). This system incorporates management by delegation concepts [1] into the Internet management framework of SNMP to facilitate the management of distributed systems [2], [3], [4]. This architecture allows a manager to delegate routine management tasks to an intermediate manager by providing a scripting MIB and language specially designed for management tasks in SNMP. This is an effective and powerful management strategy for battlefield networks which are expected to have tens of thousands of nodes. Our paper at the First ATIRP Annual Conference [5] described the motivation for and the basic principles behind the spreadsheet-based architecture and presented the structure of a proposed implementation for SHAMAN. In a subsequent paper at the Second ATIRP Annual Conference [6], we described the application of the SHAMAN system to the problem of location management for mobile nodes in a battlefield network. Both the prototype implementation of SHAMAN and a demo of the location management application were demonstrated at that conference.

Network management script delegation as the major means of Management by Delegation (MbD) has now been widely accepted by the network management research community and the industry. Many scripting frameworks have been proposed. Standards activities include DISMAN Scripting MIB [7] for SNMP management and Command Sequencer [8] for OSI management. Research prototypes other than SHAMAN include AMO [9] and GAS [10] for TMN management. As the result of these scripting framework activities and, more importantly, the increasing number of network management applications and the demand for them, more sophisticated distributed network management scripting applications are emerging. This has raised the important issue of debugging these scripting applications. Until now, the issue has been largely ignored by the network research community.

Prepared through collaborative participation in the Advanced Telecommunications/Information Distribution Research Program (ATIRP) Consortium sponsored by the U.S. Army Research Laboratory under the Federated Laboratory Program Cooperative Agreement DAAL01-96-2-0002. The U.S. government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

Management scripting applications are distributed applications. People have been working on the problem of distributed program debugging for many years and have proposed many solutions using various approaches and techniques. Of these techniques, simulation replay [11], instant replay [12], and event-based behavior modeling [13] are the most prominent ones. It is natural that the first step towards solving the problem of debugging management scripting applications is to see if we can adopt some of these solutions. In this process, it is important to find out the peculiarities of the management script applications, and evaluate the solutions accordingly.

Another important issue is finding a good debugger architecture. Since the scripting framework has many functions needed for debugging, we think it is beneficial to have a debugging architecture integrated with the scripting framework in order to limit function duplications.

The purpose of this paper is to define the problem of debugging management scripting applications and evaluate the approaches and techniques used for general distributed program debugging in the context of management scripting applications. We propose a simple approach for debugging management scripting applications. We also propose a debugger architecture which is integrated with the scripting framework. It is our hope that this architecture will allow better and easier debugging for management applications such as those that are likely to be developed for battlefield management.

II. Role of Scripting in Network Management Applications

Delegating scripts is the major means used to transfer management functions dynamically from managing systems to managed systems in order to take advantage of the increased computational power in the network elements and decrease pressure on network management centers (NMCs) and network bandwidth. Generally, Script Delegation works in the following way: a *delegation manager* downloads a set of *management scripts* which describes its desired management actions to a *delegation agent* at a remote location, and asks the scripts to be executed there. An *interpreter* in the agent then executes the scripts, and the result is conveyed back to the manager. This management scheme is illustrated in Figure 1. A *scripting framework* is used to provide mechanisms and an environment to make this management scheme possible. We describe briefly the essential components of a scripting framework in the following paragraphs. The interested reader may refer to [14] for a more detailed discussion. These components are: *script management*, *scripting language* and *interpreter*, and *script execution environment*. Scripting management deals with the following aspects of the framework: delegation roles, script administration, script transfer, and script execution monitoring and control.

The players in the scripting framework are *delegation managers (DM)* and *delegation agents (DA)*. The DM is the delegator of the scripts; the DA is responsible for the execution of the scripts on behalf of the DM. DMs and DAs have many-to-many relationships: a DM may delegate scripts to multiple DAs; a DA may execute scripts delegated from multiple DMs. The DM and DA may or may not be the management framework manager and agent respectively, although, in an integrated environment, it is very likely that DM and DA are at the same time the management framework manager and agent.

A *management script* is a set of instructions written in a scripting language, and specifies a management task; the interpretation of the script will carry out that task. However, sometimes in order to specify one management task, a set of scripts may be needed; in this case, the interpretation of the set of scripts will carry out the management task.

A well-designed scripting framework should provide good *script administration* mechanisms to avoid introducing new management problems. In order to achieve this, scripts may be assigned *names* for identification purposes; different scripts with the same name may be assigned different *versions*. Moreover, *access control* mechanisms may be provided for script access and execution.

Script transfer is the physical transportation of management scripts from a DM to a DA via some transfer mechanisms. Repeatedly executed scripts should be stored at the DAs when they finish execution. Script execution results may need to be stored locally and later processed by other scripts; or they may need to be transferred back to and examined by the DM. Results should be structured appropriately to facilitate the processing by the DMs or other scripts.

Delegated management tasks need management themselves. *Script execution monitoring and control* deals with the issues related to the monitoring and control of the progress of the delegated management tasks. The *execution environment* provides services for script executions: *translation service* translates the scripts to a required form before they are executed; *management information access service* provide MIB access and communications support; *execution service* provides multi-threaded execution, synchronization, and other services related to the execution of scripts; *error handling service* provides both static and runtime error handling.

The *Scripting Language (SL)* is defined as the language in which a management script is expressed. The use of the word “scripting” does not necessarily mean that the scripts are interpreted; actually, they could be compiled and directly executed on the tar-

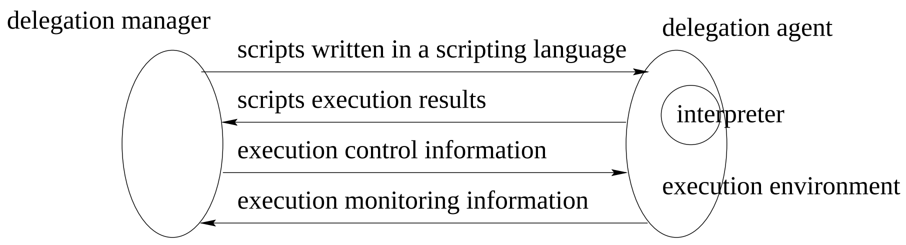
delegation manager
delegation agent
scripts written in a scripting language
scripts execution results
interpreter
execution control information
execution environment
execution monitoring information

Fig. 1. Script Delegation

get machine. We use the word "scripting" to refer to the language's characteristics of being highly capable and expressive, extensible, portable, interactive, and easy to debug.

A *management scripting application* is a distributed network management application which is composed of distributed processes and functions including scripts and their scripting framework, management managers and agents and their management framework, and other non-standard network management processes and protocols. All these processes and functions cooperate to accomplish the distributed network management task the application is written for.

III. Distributed Program Debugging

Most distributed program debugging techniques are based on or can be found in sequential program debugging. The solution for sequential program debugging is well established. The classic approach to sequential program debugging is called *cyclical debugging*. To debug using this approach, a sequential program may be executed multiple times. Because of the deterministic nature of sequential programs, the same behavior of the program is guaranteed to repeat with each execution. During the executions, the user is free to stop the program execution to examine the program state by using breakpoints, to single-step the program, or to do other things to help debugging.

As compared to debugging sequential programs, several difficulties arise in debugging distributed programs. This is because of the following characteristics of distributed programs: multiple asynchronous processes; multiple processors; variant communication delays; and non-existent or undefinable global state. Because of these characteristics, *race conditions* may exist in executing distributed programs causing *non-deterministic behavior*, and repeated executions of a distributed program may not yield the same results. Therefore, it is more difficult to apply the cyclical debugging technique in debugging distributed programs.

A few approaches have been proposed to solve this problem [15]. The simplest approach is to take snapshots of an execution of the distributed program and analyze the trace after the execution ends. The advantage is that only one execution is needed. Actually, an even simpler approach is to just display the execution at run time without actually recording the snapshots. Since successive executions may not reveal the same erroneous behavior, or any erroneous behavior at all, the disadvantage of this approach is that all information needed to debug the program must be collected during one execution. The amount of information tends to be very large and is difficult for users to sort through.

The most important approach is to make it possible to apply the cyclical debugging technique to distributed programs. The key idea of this approach is to deterministically replay an execution. In order to do this, certain amount of information must be collected during the initial execution which is called the *monitoring phase* or the *recording phase*. This technique is therefore called *record-replay* technique.

Depending on how the replay is to be carried out, there are two different techniques. One technique, which we call *simulation replay* [11], involves replaying only a subset of all processes and simulating the others for the purpose of replaying this interesting subset. In order to do this, during the monitoring phase, all interactions between the processes must be recorded together with the contents of the interactions. During the replay phase, only the interesting processes are replayed. The interactions between the processes are simulated in the sense that they are not actually carried out. The advantage of this technique is its ability to only replay the interesting processes. The disadvantage is that the amount of information that needs to be recorded is large, since we must also record the contents of each interaction between the processes.

The other technique is called *instant replay* [12]. The difference between this one and the simulation replay technique is that only the interactions between the processes are recorded; the contents of the interactions are actually reproduced during the replay phase. Thus, instant replay achieves the goal of only recording enough information in order to deterministically replay a distributed program. Therefore, the advantage is that data recorded in the monitoring phase is significantly reduced. The disadvantage is that all processes must be re-executed in order to get the contents of the interactions.

The third approach is *event-based behavior modeling* [13]. In this approach, the execution of a distributed program is seen as a sequence of events. Models are defined which specify the expected behavior of the program or the erroneous behavior of the distributed program using a modeling language. The models are then used to check against the event flow, or in other words, the actual behavior of the program. The results can be used to either help find bugs or they can be used to control the debugging activities (such as breakpointing the execution). This approach can also be used in debugging sequential programs. Since event is a widely adopted concept, the advantage of this approach is that it makes the debugging tools highly adaptable. The disadvantage is that since it is a rather formal approach and usually requires the user to learn a modeling language, it tends to be more difficult to learn and use in practice. It also requires the user to have a pretty good understanding of the system behavior in order to be able to write behavior specifications.

Other approaches which we will not introduce here include static analysis of the distributed programs and visualization of the distributed programs in time-process diagrams and animation.

An important issue in distributed program debugging is the debugger architecture design and the controlling of the debugging activities. Figure 2 shows a distributed debugger architecture. The debugger console is the host the debugger user is at and is the place where debugging commands such as single stepping a remote process, stopping or resuming all remote processes are issued. The central debug function provides an interface to monitor, control, and coordinate the remote debug functions. The GUI may provide multiple windows for viewing multiple remote processes. The communication interface implements a protocol for the exchange of debugging messages among the console and the remote debug functions. The remote debug functions are agents of the controlling debug function. They are used to collect information such as those collected in the monitoring phase of the record-replay debugging, and control the actual execution of the remote processes such as carrying out breakpoint or single step requests of the central debug function. On a system such as a multitasking system or timesharing system, the remote debug function may not have direct control of the system hardware such as CPU to do things like setting breakpoints; the actions must go through the operating system. The operating system needs to provide APIs specifically designed for debugging.

One important aspect of debugging distributed programs is how the actual debugging activities are coordinated among the distributed debugging functions. Questions such as "when a breakpoint is encountered in a process, should the other processes be stopped?" are difficult to answer but must be dealt with in order to produce a useful distributed debugger.

IV. Debugging Distributed Network Management Scripting Applications

Among the components of a management script application, the only component we want to debug is the delegated script. The other components are interesting but are not considered debugging targets. Thus, we define our goal of debugging network management scripting applications as *debugging the logical and performance bugs in the distributed scripts.* More specifically, we do not try to debug bugs in the scripting framework, faults occurring in the network, or problems in the delegation managers and management agents.

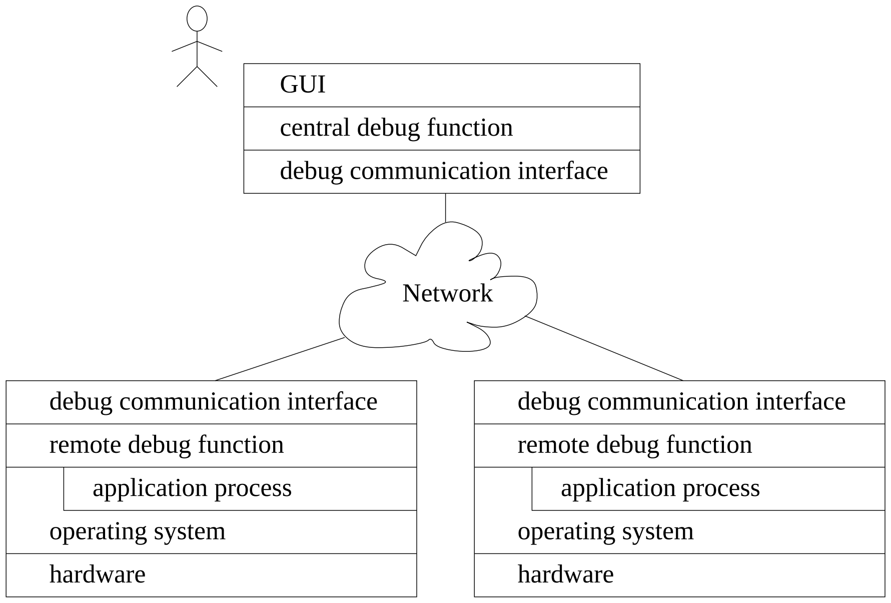

Fig. 2. Distributed Debugger Architecture

From this definition, we can characterize our debugging as *distributed embedded system debugging*. Our embedded system is comprised of the distributed scripts; the system's environment is the scripting framework, the delegation managers, the management agents, and the networking environment. It shares several characteristics of general embedded systems:

• Timer-dependent operations. For example, a script may be executed every five minutes to find how many clients are connected to our HTTP server.
• The environment is constantly changing due to some asynchronous processes not controllable by the embedded system. For example, a MIB located on an agent is constantly changing, or a manager may be executing some other related applications.
• Asynchronous environmental events. For example, managers may send requests, and agents may send notifications to the scripts.

In our embedded system, the sensors are manager requests, polls of the the management agent, and agent notifications. The actuators are management operations and the scripts issued to the management agents. Please note that the term embedded system usually has some hardware flavor, but our embedded system is a pure software one.

In order to apply the general distributed program debugging solutions we introduced in section III to scripting applications, we briefly evaluate these solutions in the new environment. Since the approach of taking snapshots of an execution of the distributed scripts and analyzing the trace after the scripts exit does not provide too much debugging help to the user, we will not discuss it further.

Of the two deterministic replay techniques, we consider Simulation Replay to be more suitable to our system. This is because of the embedded nature of our scripts. As we have pointed out, some of the processes in the whole application belong to the environment of our embedded system. They expose asynchronous behavior which is beyond the control of our debugger. For example, due to the nature of network management agents, it is not practical to try to control their operations for the purpose of debugging the scripts. Also, it is not an easy task to try to instrument the managers for the same purpose. We cannot use instant replay which does not record the contents of the interactions, and which requires total participation and control of all processes, including the manager and agent processes. We can only use simulation for the managers and the agents.

On the other hand, total record of all interaction messages requires great system resources. The instant replay technique may be useful to alleviate the problem but can be only used for interactions between scripts or any other processes over which the debugger has good control.

Since events are so fundamental to network management, the event-based behavior modeling approach is potentially a good candidate for debugging both distributed scripts and even the whole application including the managers and the agents. It also serves as a good method for network fault management. Further, it may serve as an excellent technique in interoperating the debugging of heterogeneous scripting framework applications. For our purpose, a good way to use this approach is to make it a complement of the Deterministic Replay approach. It can be used as a means to specify predicates for breakpoints to control script executions. Thus when a certain pattern of events appears, the debugger can take certain actions such as stopping the script execution, and allowing the user to examine the states of the scripts and the scripting framework.

V. Conclusions

We have shown in this paper that the management scripts are distributed embedded applications. Of the established solutions for the general distributed system debugging, simulation replay is the most suitable approach for debugging management scripts, and we have shown how it is also used to deal with the peculiar situations in the network management applications such as time-dependent operations. We also conclude that a good debugger architecture should be integrated with the scripting framework. This work has important application to the management of battlefield networks where scripting is likely to play a major role.

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

REFERENCES

[1] Y. Yemini, G. Goldszmidt, and S. Yemini. Network Management by Delegation. In I. Krishnan and W. Zimmer, editors, *Integrated Network Management II*, pages 95–107. North Holland, Amsterdam, 1991.

[2] P. Kalyanasundaram, A.S. Sethi, and C. Sherwin. Design of A Spreadsheet Paradigm for Network Management. In *Proceedings of the 7th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.

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

[4] A.S. Sethi, P. Kalyanasundaram, C. Sherwin, and D. Zhu. A Hierarchical Management Framework for Battlefield Network Management. In *Proceedings of MILCOM '97*, Monterey, CA, November 1997.

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

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

[7] David Levi and Juergen Schoenwaelder. *Definitions of Managed Objects for the Delegation of Management Scripts*, March 1997. Work in progress (Internet Draft: draft-ietf-disman-script-mib-01.txt).

[8] International Organization for Standardization. *ISO/IEC DIS 10164-21, Command Sequencer*, 1995.

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

[10] I. Yoda, H. Tohjo, and T. Yamamura. Interpreter Language-Based TMN Agent Systems. In *Proceedings of the 7th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.

[11] R. Curtis and L. Wittie. BugNet: A Debugging System for Parallel Programming Environments. In *Proceedings of the Third International Conference on Distributed Computing Systems*, pages 394–399, 1982.

[12] T. LeBlanc and J. Mellor-Crummey. Debugging Parallel Programs with Instant Replay. *IEEE Transactions on Computers*, C-36(4):471–481, April 1987.

[13] P. Bates. Debugging Heterogeneous Distributed Systems Using Event-Based Models of Behavior. *IEEE Transactions on Computer Systems*, 13(1):1–31, February 1995.

[14] D. Zhu, A. Sethi, and P. Kalyanasundaram. Towards Integrated Network Management Scripting Frameworks. In *Proceedings of the 9th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, Newark, Delaware, USA, October 1998.

[15] C. McDowell and D. Helmbold. Debugging Concurrent Programs. *ACM Computing Surveys*, 21(4):593–622, December 1989.