55
Delegation Agents: Design and Implementation

Motohiro SUZUKI, Yoshiaki KIRIHA, and Shoichiro NAKAI
C&C Research Labs., NEC Corp.
1-1 Miyazaki 4-Chome, Miyamae-ku, Kawasaki, Kanagawa, 216, Japan. Tel: +81-44-856-2314. Fax: +81-44-856-2229.
E-mail: motohiro@nwk.cl.nec.co.jp, kiriha@nwk.cl.nec.co.jp, nakai@nwk.cl.nec.co.jp.

Abstract

We have developed a management agent that adapts the delegation concept to achieve efficient distributed network management. In conventional delegation, a network management operator details management operations in an operation-script that describes management operation flow and provides such network management functions as event management and path tracing, in the form of function objects. The operator sends this script to agents to execute. In our approach, the operator sends only a script skeleton describing management operation flow alone; function objects are built into the agents themselves. This helps keep management traffic low. Each function object contains three operational objects: enhanced, primitive, and communication. Each enhanced operational object (EOO) provides a script skeleton with a network management function. A primitive operational object (POO) provides an EOO with managed object (MO) access functions. A communication operational object (COO) provides an EOO with a mechanism for accessing the functions of other remote EOOs. This design increases the efficiency of the delegation approach as applied to network management systems. We have tested our design by applying it to path tracing and found that it works well enough to suggest the feasibility of applying it to such other distributed network management functions as connection establishment and release, fault isolation, and service provisioning.

Keywords

distributed network management, delegation agent, scripting language

1 INTRODUCTION

With the explosive growth in the size of networks linking small and high-performance computers, it has become increasingly difficult for a single centralized manager to

A. A. Lazar et al. (eds.), *Integrated Network Management V*

manage entire networks. In current standardized management protocols, such as the common management information protocol (CMIP) and the simple network management protocol (SNMP), an operator must manage huge quantities of information and must control network elements (NEs) with such primitive functions as get, set, create, delete, and action. This results in an excessive processing load and extremely high manager-agent communication overheads. This weakness is particularly pronounced in the execution of network management functions that require distributed processing spread over several agents, e.g. the path tracing function proposed by the Network Management Forum (NMF) (NMF, Forum014). The function of this path tracing is to search for the termination point managed object (MO) instances which make up a specified network connection, and this search will be spread over several agents. Consequently, tracing a virtual path (VP) in an ATM network, which requires the large frequency of accessing MOs in the individual agents, causes very high manager-agent communication overheads.

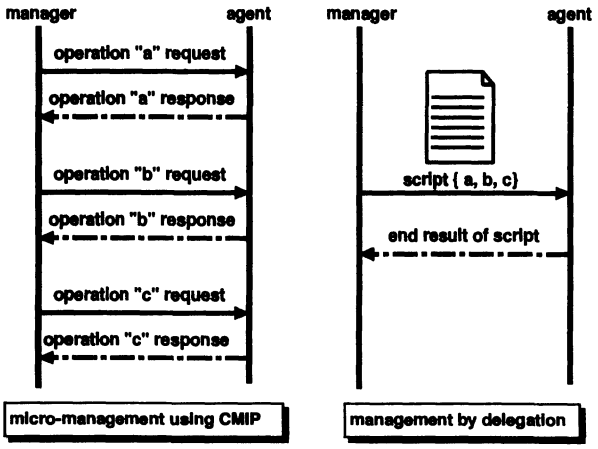

Figure 1 Micro-management and management by delegation.

To cope with this problem, one recently reported concept of distributed management, "management by delegation" (Yechiam, 1991), appears to be especially promising. In this concept, functions delegated to agents are specified in an operation-script that describes a management operation scenario in a system or scripting language. As Figure 1 illustrates, this drastically reduces the need for communication between manager and agent. Additionally, much of the information processing traditionally performed by the manager can now be handled by the agent.

In the global analysis of the concept reported in (Maria-Athina, 1996), delegation is categorized into two types: "static" and "dynamic." In static delegation, agent functions are predefined and cannot be changed or added to during its running time. In DEAL (Simon, 1996), for example, agents simply have the function of translating structured query language (SQL) operations into such SNMP operations

as get, set and get-next. While static delegation may be suitable for this type of task, there are other types for which it is not. In monitoring remote networks, such as RMON MIB (RFC1271, 1991), for example, each time a new MO definition is added to the environment, the operator must re-program and re-compile the application to deal with the new MO definition. That is, since static delegation agents cannot dynamically extend their functions, they are unable to cope with the new requirements or unexpected situations that may arise during their running time.

Dynamic delegation is much more flexible. With it, an operator can provide agents with an operation-script that describes the implementation of any new functions to be added. In this way, the operator can respond to new requirements and resolve unexpected situations. In the application monitoring of remote networks described above, for example, an operator needs only to send to the agent a new operation-script that describes the new functions needed to deal with the new MO definition.

While much discussion has been reported on the concept of applying dynamic delegation to network management, however, little actual design and implementation has been conducted, particularly in comparison to the great amount that has been conducted with respect to static delegation. In our study, we have devoted the bulk of our efforts to these two needed areas: actual design and implementation.

In Section 2 of this paper, we describe the special requirements of "network management by delegation." In Section 3, we propose a new management agent architecture for satisfying these requirements. In Section 4, we describe the implementation of our design, and in Section 5, we summarize the study.

2 DESIGN REQUIREMENTS

In this section, we discuss the requirements for applying dynamic delegation to network management.

Format independence: Operation-scripts in conventional delegation agents are written in a system language, such as C or C++, and must be at least partially written to suit the specific format in which each individual agent is implemented. Consequently, in order to use an operation-script that has been written for one format in a different format, a network management operator must appropriately modify the format-dependent part of it, i.e., that part which depends on the management application programming interface (API) implemented in the agent, such as OSIMIS (George, 1995) and XMP (X/Open). In order to avoid all the problems that this creates, operation-scripts should be format independent.

Agent-agent communication: When a network management function requires distributed processing spread over several agents, these agents must be able to communicate with one another without adding significantly to the operator's processing load or to manager-agent communication overheads.

Secure execution: When a single operation-script is sent to a number of different agents, it may happen that operations permissible for one agent are not permissible for another, and a control mechanism is needed to prevent agents from executing operations that are, for them, illegal.

Common process sharing: When the same single process is described in a number of different operation-scripts that are to be executed simultaneously, a considerable amount of waste will be incurred. Furthermore, the operator will have had to write the processes in each of those individual scripts. Efficiency requires the elimination of this overlap.

3 DELEGATION AGENT ARCHITECTURE

In this section, we describe the agent architecture we have developed for use with dynamic delegation. In the architecture, we propose a new operation-script format and execution mechanism that helps satisfy the requirements described in Section 2.

3.1 The script skeleton and function objects

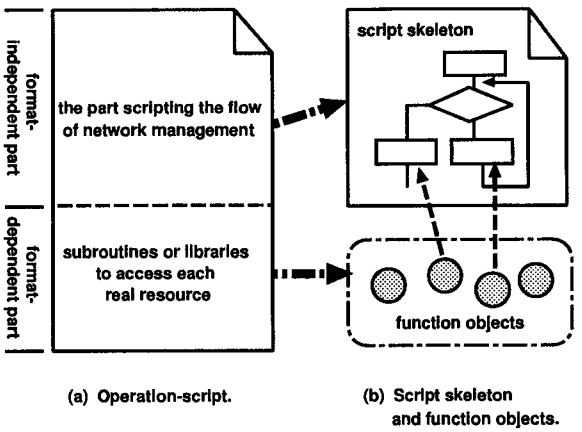

Figure 2 Operation-script, script skeleton, and function objects.

We propose here a new operation-script format that is independent of individual agent implementation formats. The operation-script now sent to an agent contains only a "script skeleton," which describes format-independent operational flow alone; function objects, which provide such network management functions as event management, path tracing, etc., are built into the agents themselves (Figure 2(b)). Additionally, to execute network management functions that require distributed processing efficiently, a function object has a mechanism for agent-agent commu-

nication. In the mechanism, to invoke functions provided by EOOs existing remote agents, a function object sends script skeletons to the agents.

A network management operator needs only to write a single script skeleton to be sent to all agents, regardless of their individual implementation formats, which helps keep network traffic from the manager to the agents low.

3.2 Operation-script execution mechanism

Figure 3 illustrates the operation-script execution mechanism. When the script processor receives a script skeleton from a manager, its execution control module checks whether all required function objects are allowed to bind with the script skeleton. This check is performed on the basis of an access control list contained in the delegation information repository (DIR). If all the objects are allowed to be bound to the skeleton, the script processor performs the binding, executes the skeleton, and sends execution results to the manager. Since the execution control mechanism checks the permissibility of any script skeleton, there is no worry of an agent's performing illegal operations.

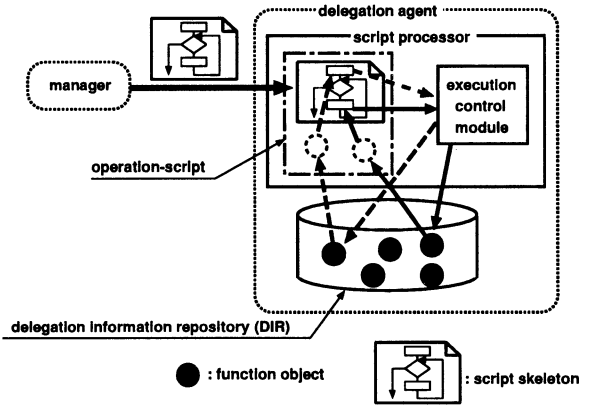

Figure 3 Operation-script execution mechanism.

The use of function objects itself helps greatly in sharing common processes. That is, sharing function objects among script skeletons reduces a considerable amount of waste in executing the skeletons. Moreover, to improve the efficiency of the script skeleton execution, the same mechanism contained in function objects is shared among the objects. In a typical example of the sharing, function objects contain the same mechanism for agent-agent communication.

4 IMPLEMENTATION

In this section, we discuss how the manner in which function objects are implemented helps to apply our architecture to dynamic delegation efficiently. Regarding the manner, to achieve concurrent MO access and provision of a simple and powerful API, we employ the concept of component-ware, in which function objects consist of three types of operational objects: "enhanced," "primitive," and "communication."

4.1 Operational objects

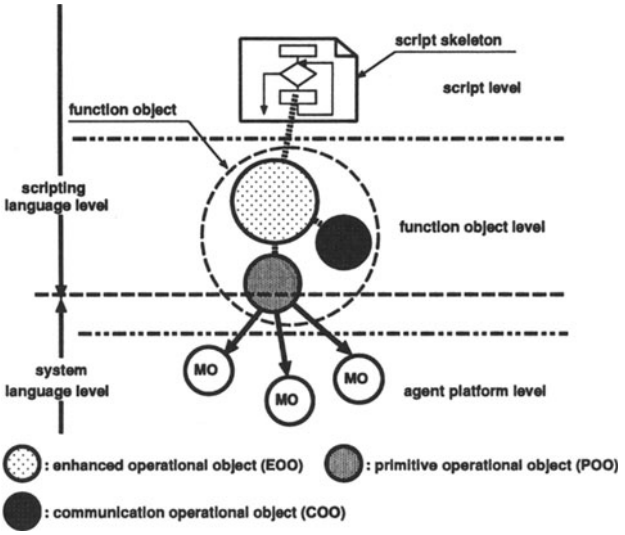

Figure 4 Function object design.

Enhanced operational objects (EOOs) contain an API for interfacing with script skeletons. Primitive operational objects (POOs) access MOs for EOOs. Communication operational objects (COOs) provide EOOs with a mechanism to achieve agent-agent communication. Figure 4 illustrates the relationships among these objects. Each function object here is composed of an EOO, a POO, and a COO. The script skeleton specifies a function provided by the EOO, and the EOO orders its POO to access all relevant MOs. The POO does this and returns their attribute values to the EOO for processing. The EOO then returns the results of this processing to the skeleton. In the following, we discuss individual objects in more detail.

Enhanced operational objects (EOOs): An EOO is capable of complex processing of MO attribute values, and it contains an API for interfacing with the script skeleton. To help locate MOs that have been specified in a script skeleton,

EOOs contain a function for translating abstract names into the data to which they correspond, which is written in a scripting language that specifies data types (**string**, **integer**, etc.), as well as into attribute labels and the distinguished names (DNs) of MOs. Abstract names are simple and easily understood by an operator. An EOO communicates with EOOs existing in the remote agents to execute network management functions that require distributed processing. To communicate with remote EOOs, the EOO utilizes a mechanism provided by COOs.

Primitive operational objects (POOs): POOs access one or more MOs according to requirements of EOOs. To facilitate easy handling of MO attributes in an EOO, a POO translates a data written in a scripting language into the corresponding data written in a system language, and vice versa. For example, a POO translates a character string that represents the DN of an MO instance into the DN-structure in C++. POOs utilize a thread mechanism to access multiple MOs concurrently in order to achieve real-time processing.

Communication operational objects (COOs): COOs provide EOOs with a mechanism for agent-agent communication. This mechanism is particularly useful in implementing network management functions that require distributed processing at the agent level. The COO first locates a remote EOO providing a required function and then sends a script skeleton invoking the required function to the remote EOO for execution.

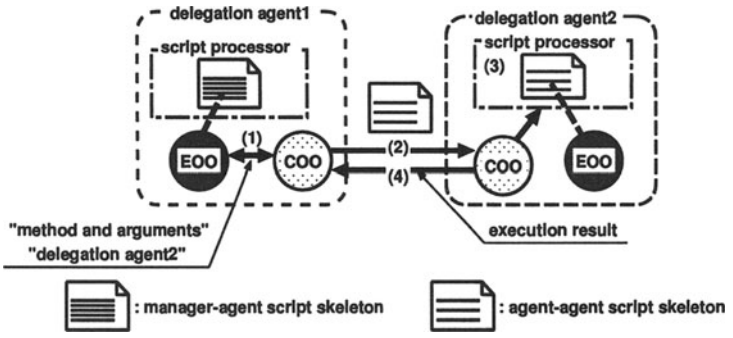

Figure 5 Agent-to-agent communication.

Figure 5 shows this process in which agent-agent communication is achieved via COOs between two delegation agents called "delegation agent1" and "delegation agent2." An EOO of "delegation agent1" first gives its COO the names of the required function and of an agent that contains it (in this case "delegation agent2"), as well as the arguments used to invoke the function (Figure 5(1)). This COO sends a script skeleton containing these arguments to the COO of "delegation agent2" (Figure 5(2)), which forwards the skeleton to the agent's script processor. The script processor locates the EOO containing the required function, executes the

skeleton with it (Figure 5(3)), and gives the results of the execution to the COO to be sent back to the COO of "delegation agent1" (Figure 5(4)).

4.2 Case study: path tracing

To test the feasibility of our delegation agent architecture, we have applied it to the task of tracing a VP trail in ATM networks. Figure 6 illustrates an example of tracing a VP trail called "vpTrail#1" between two ATM NEs called "node_A" and "node_B." A manager sends a script skeleton invoking the function for tracing "vpTrail#1" to "delegation agent1" managing "node_A." In the skeleton, an operator describes the abstract name specifying the required VP trail (in this case "vpTrail#1"). In "delegation agent1," "vpTrail#1" is translated into the DN of the VP trail termination MO instance (e.g. the *vpTTPBidirectional*-MO instance (Alex, 1996)) to enable MO access. If tracing the VP trail requires invoking the path tracing function provided by the other delegation agent (in this case "delegation agent2" managing "node_B"), "delegation agent1" sends a script skeleton for tracing "vpTrail#1" to "delegation agent2." In "delegation agent1," the skeleton is automatically created with the DN of an MO instance for initiating MO access in "delegation agent2." Then, "delegation agent2" sends the execution result of the skeleton to "delegation agent1," and "delegation agent1" sends the tracing result back to the manager.

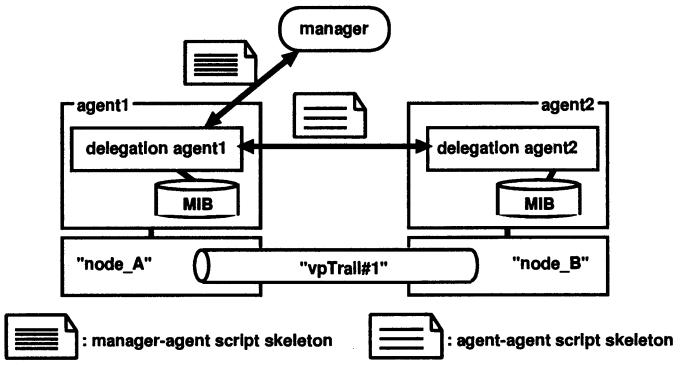

Figure 6 Tracing a VP trail.

Currently, we have implemented the operational objects and script skeletons in Java (Ken, 1996) except the MO access portion of the POO, which is written in C++. We used TCP/IP socket libraries to transfer script skeletons. The results of the implementation show that sending only a script skeleton to an agent enables the management traffic between a manager and an agent to be kept low, and it works well enough to suggest the feasibility of applying our design to other distributed network management functions as well as path tracing.

5 CONCLUSION

This paper has proposed a new architecture for management agents that is designed to achieve efficient distributed network management. In this architecture, which employs the dynamic delegation concept, a network management operator describes management operations in an operation-script, and the script is sent to an agent to execute. Adopting the delegation concept helps operators cope dynamically with any situation in network management.

To implement the dynamic delegation concept efficiently in network management systems, we combine the use of a script skeleton with that of function objects. A script skeleton is independent of any agent implementation format and describes management operation flow. Function objects provide network management functions and are stored in the agent. To help keep management traffic low, only a script skeleton is sent to an agent, where it is dynamically bound with operational objects in order to execute a complete operation-script. In designing function objects, we have employed the concept of component-ware, in which function objects consist of multiple operational objects. This design supports efficient distributed network management: it keeps management traffic low and extends agent functions dynamically.

REFERENCES

Network Management Forum. Application Services: Path Tracing Function (Forum 014).

Yechiam, Y., German, G. and Shaula, Y. (1991) Network Management by Delegation. 2nd International Symposium on Integrated Network Management.

Maria-Athina, M. and Gabi, D. (1996) Delegation of functionality: aspects and requirements on management architectures. 7th Distributed Systems: Operations & Management.

Simon, Z., Michel, L. and Jean-Pierre, H. (1996) DEAL: delegated agent language for developing network management functions. 1st International Conf. and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology.

RFC 1271 (1991). Remote Network Monitoring Management Information Base. IAB.

George, P., Kevin, M., Saleem, B. and Graham, K (1995). The OSIMIS platform: making OSI management simple. 4th International Symposium of Integrated Network Management.

X/Open Preliminary Specification. System Management: Management Protocols API. X/Open Company, Ltd.

Alex, G. (1996). Access Network Management Modeling. IEEE Communications Magazine, March, 62-72.

Ken, A. and James, G. (1996). The Java Programming Language, Addison-Wesley Company.

Motohiro SUZUKI received his B.E. and M.E. degrees in information systems engineering from Osaka University in 1992 and 1994, respectively. He joined NEC Corporation in 1994 and is now a member of the Network Research Laboratory, C&C Research Laboratories. He has been engaged in the research and development of network management systems.

Yoshiaki KIRIHA received his B.E. and M.E. degrees in electronic communication engineering from Waseda University in 1985 and 1987, respectively. He joined NEC Corporation in 1987 and is now a member of the Network Research Laboratory, C&C Research Laboratories. He has been engaged in the research and development of network management systems and distributed artificial intelligence systems.

Shoichiro NAKAI received his B.E. and M.E. degrees from Keio University in 1981 and 1983, respectively. He joined NEC Corporation in 1983 and has been engaged in the research and development of local area networks, distributed systems, and network management systems.