TOWARDS INTEGRATED NETWORK MANAGEMENT SCRIPTING FRAMEWORKS [1]

Dong Zhu, **Adarshpal S. Sethi**
Department of Computer and Information Sciences
University of Delaware, Newark, DE 19716
email: (dzhu, sethi)@cis.udel.edu

Pramod Kalyanasundaram
Lucent Technologies
Room 4F-335, 101 Crawfords Corner Road, Holmdel, NJ 07733
email: pramodk@dnrc.bell-labs.com

Abstract

Management scripts are being used as the major means to realize the powerful concept of management by delegation. Many script delegation frameworks are proposed and experimented with for various network management standards, notably, the SNMP, OSI, and CORBA. More proposals and research are underway. The major problem with these scripting frameworks is that most of these frameworks are, in a sense, "quick fixes" aiming at rapidly introducing remote scripting capability to existing management frameworks; therefore, true integration between the two are not achieved. We argue that an integrated scripting framework can provide more power and ease of use to the network managers and applications. In this paper, we explore various ways to integrate the scripting and management frameworks. This is based on our decomposition of the management information model, and a definition of the functional architecture of the general scripting framework. We also present a perspective on different levels of integration.

Keywords: Integrated Scripting Framework, Management Information Model, Object Information Model, Object Relationship Model, Object Distribution Model, Scripting Language

1 Introduction

The evolution of network management has resulted in the development of various network management standards. The two most widely deployed standards are the Internet management standards that use the SNMP protocol [CMRW96], and the ISO/CCITT OSI Systems Management standards, using the CMIS/CMIP [IS990a, IS990b] services and protocol. Now, with the development and popularity of the distributed object technologies,

[1] 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.

we are seeing a growing effort in trying to use these technologies to ease the task of network management. Of these distributed object technologies, Object Management Group's CORBA [COR95] is among the most promising ones.

Management by Delegation (MbD) [YGY91] is a well-known strategy for implementing hierarchical management, and management scripts are the major means used today to delegate management functions to lower level managers. There have been many research and standardization activities aiming at providing scripting frameworks for the management standards. DISMAN Scripting MIB [LS97] is being standardized for SNMP management. SHAMAN [KSSZ97] has a novel design for SNMP management using a spreadsheet abstraction of the MIB. Command Sequencer [Com95] is being standardized by ISO/CCITT for OSI management; AMO [VPK97] and a generic interpreted language-based agent system (GAS) [YTY96] have been designed for TMN [TMN92] management. CMIS++ [AC96] is proposed for the extension of CMIS scoping and filtering expressive power. We have not seen any scripting frameworks for CORBA-based network management, but it will not be long before we see some research conducted in this area, since there is a growing interest in using CORBA for network management.

The major problem with these scripting frameworks is that most of these frameworks are, in a sense, "quick fixes" aiming at rapidly introducing remote scripting capability to existing management frameworks; therefore, true integration between the two are not achieved. For instance, many scripting frameworks employ Tcl-based scripting languages, but the management information models these languages manipulate are not integrated with Tcl's data types and data structures. In our opinion, an integrated scripting framework can provide more power and convenience to the network managers and applications.

Our main purpose in this paper is to explore various ways to integrate the scripting framework and management framework. In order to achieve this goal, first, we have decomposed the management information model into three closely related component models, i.e., the Object Information Model, the Object Relationship Model, and the Object Distribution Model. We have also identified the functional architecture of the general scripting framework, which consists of the Script Management, the Scripting Language, and the Script Execution Environment. We then try to find out the potential ways that the component management models and the scripting framework can integrate with each other. We also present a viewpoint on different levels of integration of the two frameworks.

The organization of the rest of the paper is as follows: Section 2 gives an overview of the Management Information Model and breaks it into three component models. Section 3 introduces the concepts in the scripting framework, and gives a functional model for it. Section 4 explores some possible ways that the scripting framework and the management framework can be integrated. Section 5 presents a viewpoint of the different levels of integration. Section 6 discusses future work and concludes the paper.

2 A Decomposition of the Management Information Model (MIM)

Traditionally, the scope of management operations has been delineated by the information model within which the management protocol functions. The Management Information

Model (MIM) is both the conceptual and concrete framework within which management information can be defined, constructed, accessed, and monitored. For our purpose, MIM is seen as being constructed from three closely related component information models, i.e., the Object Information Model, the Object Relationship Model, and the Object Distribution Model.

The *Object Information Model (OIM)* defines the *object* aspect of the MIM by which we refer to the capabilities of the model to define objects used to represent management information, the naming of these objects, and the ability to execute various types of operations on these objects. More specifically, OIM is concerned with the following issues: What data types may be used to represent management information? Is the OIM object-oriented? If yes, what object-oriented features does it use? What properties does an object have? How are objects and/or object classes defined? How are objects created, deleted, modified, accessed, and monitored? How are objects and their properties named?

The *Object Relationship Model (ORM)* defines the *object relationship* aspect of the MIM by which we refer to the capabilities of the model to define relationships between objects, construct MIBs, and to provide mechanisms for users to manage these relationships. Since a managed object is the representation of some management information, and the collection of a set of managed objects can be seen as a Management Information Base (MIB), the ORM actually specifies the logical structure of the MIB. Besides its capability to construct a MIB, ORM may also provide other mechanisms so that other kinds of relationships, such as a reference relationship, can be established between objects by the user. More specifically, the ORM is concerned with the following issues: What relationships are allowed between objects? How are these relationships represented? How can users establish, access, monitor, traverse, and manipulate relationships? Does the ORM allow dynamic construction of the MIB? How flexible are the MIBs? How are relationships related to the Object Information Model? How does the ORM affect object naming and addressing? How do the ORM and object creation/deletion interact?

The *Object Distribution Model (ODM)* specifies the *distribution* aspect of the MIM by which we refer to the capabilities of the model in manipulating objects in a distributed environment. More specifically, ODM is concerned with the following issues: Are the locations of the objects transparent to the user? In other words, does the naming or addressing of the objects allow users to be unaware of the locations of the objects? When location awareness is needed in management, how is that provided? How are relationships established and accessed in the distributed model? Can relationships be established and accessed in the same way as in the non-distributed case? How are objects created and deleted in a distributed model? How are objects copied and moved across nodes? Are objects allowed to move with their state information? Can objects move themselves?

From SNMP to OSI, the OIM of the MIM is greatly enhanced. SNMP does not use an object-oriented model, whereas OSI does. From OSI to CORBA, the OIM of the MIM remains mainly the same. Both use object-oriented models. However, since OSI is defined specifically for network management, it has a few management-oriented features which are not present in CORBA.

The ORM and ODM are not significantly enhanced from SNMP to OSI. The relationship models of both SNMP and OSI are weak compared with CORBA. Neither SNMP nor OSI uses a distributed model. CORBA, on the other hand, has a very general and pow-

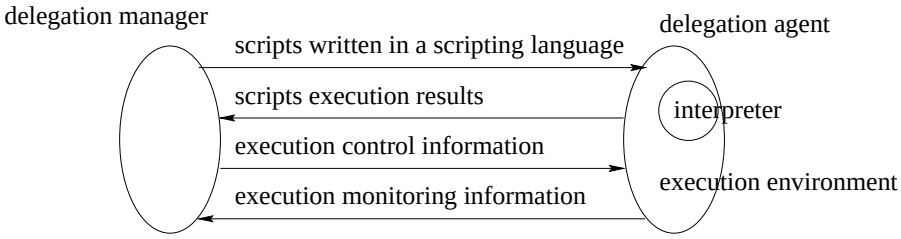

Figure 1: Script Delegation

erful ORM. Besides modeling relationships as CORBA objects, it has also defined a service to manipulate graphs of objects. CORBA also uses a distributed object model in which the distribution of the objects is transparent to the object clients. Further, the objects are allowed to be moved along with their state information.

3 A Functional Model for the Management Scripting Framework

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. It is intended to solve two major problems of the current management paradigms. One problem is related to the limitations of centralized network management paradigms such as SNMP and OSI. Another is related to the limitations of static delegation approaches as exemplified by RMON [Wal95] and OSI's Systems Management Functions (SMFs) [OSI93, to name one]. The results are high level of scalability due to the distribution of management functions, increased flexibility through *dynamic delegation*, and increased survivability of distributed systems due to increased autonomy of managed systems.

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* should not only provide mechanisms and an environment to make this management scheme possible, it should also provide great convenience to the framework users in the entire process. We have identified the essential components of a scripting framework and will discuss them in this section. These components are: *script management*, *scripting language* and *interpreter*, and *script execution environment*.

3.1 Script Management

Here, "script management" refers to script administration, script and script execution result transfer and storage, 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 *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.

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 executions. 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. The major issues related to script and script execution result transfer and storage are:

- What is the *script transfer model* used? There are two models depending on who is responsible for the script transfer: if it is the DM, the model is called *push model*; or if it is the DA, the model is called *pull model*.
- Who initiates the script transfer? Transfer could be initiated by a human manager or by some automatic process triggered by events.
- What protocol is used in transferring the scripts? Numerous kinds of protocols have been suggested and used including management protocols such as SNMP and CMIP, file transfer protocols such as FTP, transport layer protocols such as TCP, and other protocols such as HTTP.
- How are the results transferred back to the manager? Issues applicable to the script transfer also apply here: What is the result transfer model used? Who initiates the transfer? What protocol is used in transferring the results?
- Are the results allowed to be stored locally and later processed by other scripts? How are the results stored?
- How are the results structured? A result could be as simple as an integer value or a network address, or it could be as complex as a list, a table, a tree, or some other more sophisticated data structures.

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. Major issues involved are:

- What are the states a script can be in during its execution? Possible states are: *ready*, *running*, *suspended*, *aborted*, *terminated*, etc.
- Does the framework allow the states and progress of script executions monitored? How is this provided?
- How is an execution initiated? A script execution may be initiated as the result of an explicit manager request asking for the execution of a script, the reaching of a pre-scheduled time, or some triggering events such as an alarm. A script execution may be initiated immediately after the script is downloaded to the DA. Other scripts

may refer to/call this script; their execution will trigger the execution of this script (We can see this as "function call" in programming languages.).

- What are the execution controls allowed? Possible controls include: initiation, suspending, resuming, aborting, signaling (or sending events to an execution), enabling, disabling, etc.
  What level of control of script execution does the framework allow the managers to do? *Coarse control* may only allow the managers to initiate or abort a script execution, while *fine control* may allow all controls listed above.
- Are pre-scheduled and periodic executions allowed? How are they specified by managers?

3.2 Scripting Language (SL) and Interpreter

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 target machine. We use the word "scripting" to refer more to the language's characteristics of being highly capable and expressive, extensible, portable, interactive, and easy to debug. The scripting language is probably the most important aspect of the scripting framework because the power that scripts may have depends mostly on it.

A SL may have all the major data and control structures as a general purpose programming language. Besides the regular data model the SL manipulates, the SL may have other data structures which are specifically provided to deal with the target MIM that the scripts manipulate. For instance, for DISMAN and SHAMAN, the target MIM is SMIv2; for AMO, it is GDMO.

The SL may also have control structures that are designed specifically for management applications. The most noticeable example is the asynchronous event processing control structures which allow the scripts to be written to process asynchronous events.

In order to allow scripts to access MIB and management protocols, SL usually provides necessary language constructs to achieve this. For instance, many Tcl-based SLs for SNMP extend Tcl by adding commands to do SNMP Get, GetNext, GetBulk, Set, etc.

In order to execute a script, a runtime environment must be provided for the scripts to access MIBs, communication protocol stack, and other management facilities. These will be discussed in the next subsection.

We can distinguish two kinds of MIB accesses: *local access* and *remote access*. Local access deals with the local MIB, while remote access deals with a remote MIB and also needs management protocol support. For remote access, the scripts act as a MF manager. The goal to achieve here is to provide in the SL the full remote access power a manager may have. One desirable goal is the transparent access to both local and remote MIBs.

Major issues involved in SL are:

- Interpreted versus compiled language. An interpreted language generally means higher portability, more interactive and easier to debug; a compiled language generally has higher performance. An interpreted script is bound to its execution environment loosely through the interpreter, while a compiled script needs to be bound to the execution environment much more tightly than an interpreted script.

- An existing general purpose scripting language with management extensions versus a newly crafted scripting language. The first approach has the merit that the developing process is quick but suffers from the fact that it is not very well integrated with the target MIM the scripts manipulate. The second approach is exactly the opposite of the first approach. Using either approach, the SL should be chosen or designed in a way so that the target MIM is easily manipulated.
- How is an existing general purpose scripting language extended to a management scripting language? Very often, this is done by adding new instructions as the way Tcl is extended. Or it may be extended by providing functions for management operations. The language has basic management information access primitives used to access the target MIM. For instance, a SL designed for SNMP would provide primitives such as Get, Set, GetNext, GetBulk, and Inform, corresponding to the SNMP PDU types.
- Does the language provide asynchronous control facilities? This is very important for network management.
- What are some of the useful management protocol accesses provided? For example, ICMP, HTTP, DNS access primitives are provided in Scotty [SL95], a Tcl extension for network management.

3.3 Script Execution Environment

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. Major related issues are:

- What is the required code form for scripts? What is the required code form for execution? If the forms are different, translation is needed before the scripts can be executed. The forms could be source code, intermediate code, or object code.
- How are scripts bound to local management instrumentation to access local management information? How can scripts access remote management information using the management protocol and services provided via the local system?
- Are simultaneous executions of scripts allowed? How is the integrity of the management system maintained? How are "race conditions" avoided?
- Are multiple instantiations of a script execution allowed? How is it achieved? The reason why this is an issue is that a mechanism is needed to identify the instantiations, and to communicate with, monitor, and control a specific instantiation. Also, if a script can generate output, different instances should write their output to different places in order to avoid interference with each other.
- Does the framework allow accounting of the resources used by a set of scripts? How is this achieved? Accounting is necessary to control misbehaved scripts which may wrongly consume a major portion of local computing power and/or network bandwidth.
- How are both static and runtime errors handled? How are errors reported to the delegation manager?

3.4 Management by Delegation (MbD): An Example of Script Delegation

In MbD [Gol95], a *delegator process (DP)* (delegation manager) sends *delegated agents (DAG)* (management scripts) to an *elastic process* (delegation agent). The elastic process runs the delegated agents as threads and provides device instrumentation and SNMP support.

The *remote delegation service (RDS)* (script management service) provides services for delegation and DP to DAG communication. Once a DAG is delegated, a *handle* (script name) is passed back to the DP using the RDS. The RDS also provides security mechanisms for DP authentication and access control of DAG to the local resources. A DP can delegate and delete DAGs using RDS-Delegate and RDS-Delete; it can send messages to a DAG using RDS-ReceiveMsg; a DAG can send messages back using RDS-SendMsg. a DP can do RDS-instantiate, RDS-Terminate, RDS-Suspend, and RDS-resume.

Elastic process *runtime environment* implements the delegation services. *Delegation protocol* is designed to provide the RDS. A database called *repository* provides the storage for DAGs. *Contoller* is used to initialize the runtime environment. *Translator* compiles source DAG code. *DPI-Mgr* allows the remote delegator process to do execution control. *IPC* supports inter-agent communications. *Scheduler* provides multitasking scheduling.

MbD allows arbitrary programming languages. Agent can be in either source code, intermediate code, or object code format. The above mentioned *translator* compiles source code DAGs.

4 Integration between the Scripting Framework (SF) and the Management Framework (MF)

The purpose of this section is to explore the possibilities of SF and MF integration. We will investigate the integration of the SF with each of the three component models of the MIM, namely, the OIM, the ORM, and the ODM.

4.1 Integrate OIM with SF

Scripts can be modeled as management information as exemplified by AMO [VPK97]. Once we have modeled scripts in this way, a lot of MF functions (MFFs) can be used to achieve many script management functions. In other words, the SF can be partially implemented by the MF. More specifically, we can identify the following integration points of the two frameworks:

- Delegation managers and agents are modeled as MF managers and agents respectively.
- Scripts and their execution results are modeled as management information and are stored in the MIB of the MF.
- Since the scripts and their results are stored in the MIB, they can be transferred using the information transport service of the MF. For instance, SNMP and CMIP can be used to transfer the scripts and results.
- Script executions are very much like the management resources whose states are changing, who can be controlled by manager-requested actions, and who can generate events; therefore, it is natural to model script executions as managed objects. Consequently,

the monitoring and control of the script executions can also be achieved through the MF mechanisms.

We can also integrate the OIM with the scripting language (SL). As we have explained in the section discussing the SF, the SL must provide at least some kind of basic facilities for the scripts to access the management information. For example, SNMP MIB access requests such as Get, GetNext, GetBulk, and Set can be provided in a SNMP SL as primitive instructions. This approach only provides a "minimal" integration of the SL and the MIM: the access functionality is easily identified in the SL as a few extra "add-ons", and no value-added functions are provided by this integration.

A better approach is to tightly integrate the MIM with the SL. For example, a MOC may be modeled as a SL object class and a MO as a SL object; operations invoked on the SL object will in turn be invoked on the corresponding MO. This approach provides to the script programmer a more unified view of the information models that the SL manipulates. In our opinion, this will make script programming easier.

The SF can be used to extend the OIM by using scripts to specify management behavior. The behavior of MOs can be specified by management scripts. This can be trivially explained as: since we can model scripts and their executions as MOs, the behavior of these MOs are determined by the scripts. However, this is not the only way scripts can be used to specify MOs' behavior. A more sophisticated and integrated way is to use scripts to specify the behavior of non-script MOs. For example, scripts may be used to specify the behavior of MO attribute accesses, so that corresponding script-programmed actions may be taken when Get or Set are applied on these attributes. Scripts may also be used to specify the actions that a MO supports. In order to achieve the above, the SF and the MF need to cooperate so that associations can be set up between MO behavior and scripts. GAS [YTY96] uses this integration approach.

4.2 Integrate ORM with SF

We can integrate the ORM with the script management functions. If we model scripts as MOs, we can establish relationships between scripts, between scripts and other entities in the SF and MF, and between other entities in the SF. Right now we have identified the following useful relationships:

- Dependency relationship between scripts and events: when an event happens, the dependent scripts are executed.
- Sequential relationship between scripts: one script can "call" other scripts.
- Concurrent relationship between scripts: scripts can run concurrently.
- Relationships can be established between script execution results. SHAMAN [KSSZ97] has given some good examples in this respect: a multi-value cell relates several results as a list; a virtual table relates results as a two-dimensional table which can be manipulated by the SSL of SHAMAN.
- Association may be built between scripts and MO behavior specifications so that MO behavior may be specified by scripts, possibly dynamically.

Similar to the approach we propose to tightly integrate OIM with the SL, we can integrate the ORM with the SL in the following way: relationships are modeled as SL data

structures; operations are defined to manipulate these data structures. One can have the following possibilities:

- A SL data structure representing a tree structure may represent a SNMP MIB tree. Operations defined to traverse the data structure may in turn traverse the actual MIB tree.
- A SL data type may be used to represent a SNMP table. Operation can be defined for the data type to manipulate the table. For instance, query operations may be defined for the table.
- A SL data type may be used to represent an OSI MIB. More powerful traversal, scoping, and filtering operations similar to those proposed in CMIS++ [AC96] may be provided.

4.3 Integrate ODM with SF

We can integrate the ODM with the script management functions. With an ODM where location transparency of MOs is provided, we may extend the feature to the management scripts: delegation managers may not be interested in where exactly scripts are and should be able to execute scripts and get the results without the knowledge of the location information. The integrated SF and MF will ensure the above to be correctly carried out, forwarding information between the managers and scripts. They may also provide copies of a script in the network and execute the copy which they consider most suitable for the management task.

Also, with an ODM like that of CORBA's, we may be able to design a SF integrated with the ODM which makes it possible for the SF to easily move and copy scripts. Further, scripts may be allowed to move themselves along with their states within the network — mobile agent — in order to carry out certain management tasks.

To integrate an ODM as powerful as that of CORBA's with the SL, the most important thing is perhaps to try to incorporate the full functionality of the ODM with the SL: the power of the SL will be inherited from that of the ODM. This would more likely include integrating into the SL location transparent MO access, moving, copying, etc.

In the situation when scripts are allowed to move with their execution status and other information, target execution environment will be responsible for restoring the execution state and be able to resume the execution.

4.4 Extending MIM using SF

A SF provides an overlaying framework over the MF, and most of the management information of the MF can be accessed from the SF. We may provide new Information models, Relationship models, and Distribution models in the SF which can enhance the MF MIM. In a sense, we can add functions in the SF at our will, since the SF is overlaying on top of the MF which is not aware of the changes. For example, we can imagine adding an OO information model and a distributed model in the SF for SNMP management, which will give us real OO accesses to SNMP management information, and object mobility. As another example, we can imagine adding a relationship model in the SF for SNMP management, which will give us dynamic MIB construction capability.

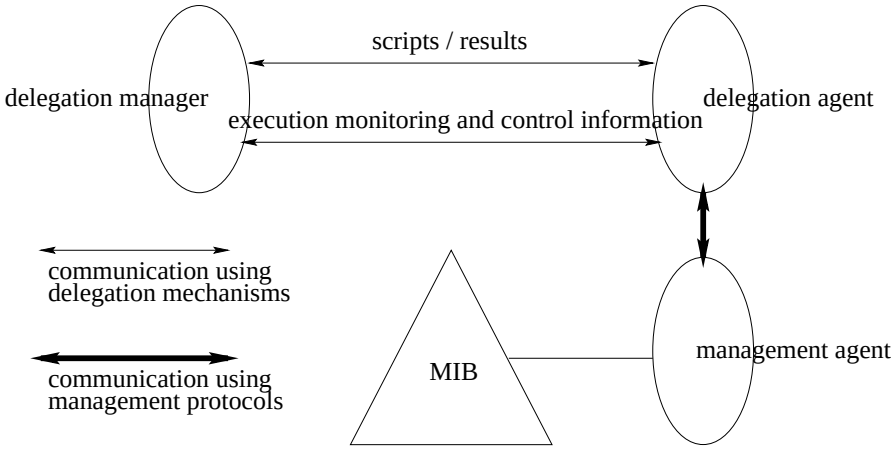

Figure 2: Minimal Integration of the Scripting Framework and the Management Framework

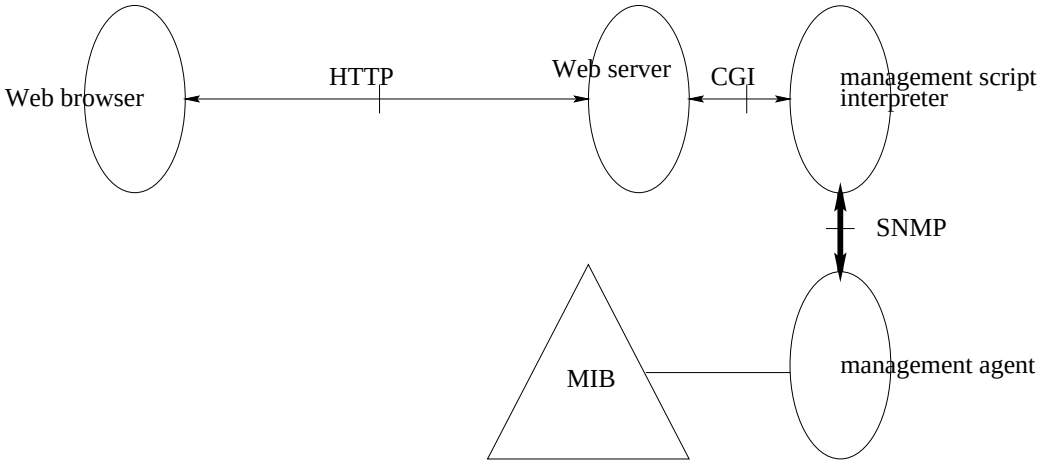

Figure 3: A Scripting Scheme using the WWW

5 A Viewpoint on Different Levels of Integration

5.1 Minimal Integration

This approach is illustrated in Figure 2. Using this approach, the SF is almost totally separate from the MF. The delegation managers and delegation agents are different from the MF managers and agents, respectively. These delegation managers and agents are free to choose the "script management" methods, especially the methods for: script transfer and storage, script execution result structuring, storage, and transfer, and script execution monitoring and control.

However, we can still identify two places in the SF where integration with MF is made. One place is the scripting language (SL) and the MIM integration, where SL is designed with the capability to access the MIM. At this stage of integration, the SL is usually minimally integrated with the MF MIM, by which we mean the SL can be used

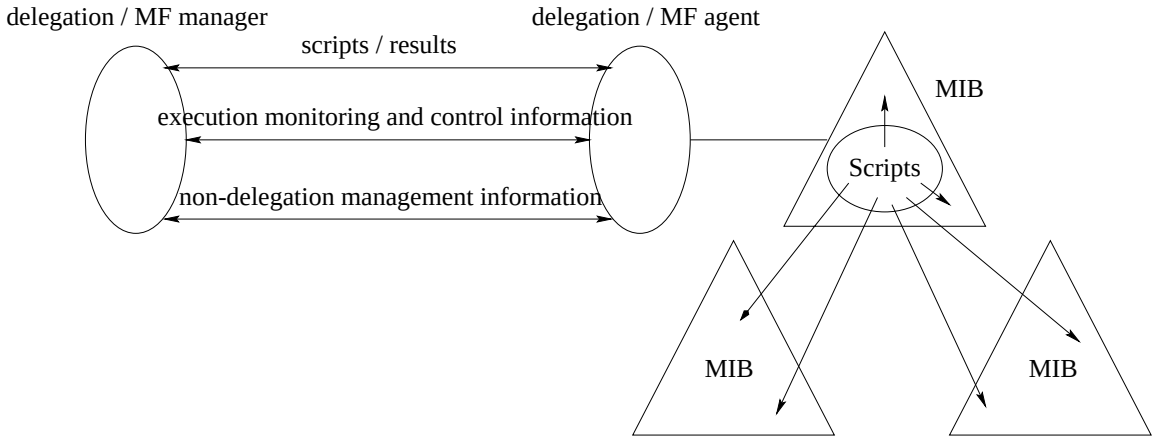

Figure 4: The Basic Integration Approach

to access the MIM, but the data structures of the SL are not well integrated with the MIM. For instance, many SNMP SFs use a Tcl-based SL, where there are a few add-on instructions to do SNMP Get, GetNext, GetBulk, Set, etc. However, there is no facility to make a variable represent an SNMP table, group, or a MIB. We think this kind of facility can greatly enhance the scripting language expressive power.

Another place is the SF delegation agent and MF manager integration — the SF delegation agent acts also as the MF manager and is therefore a gateway between the SF and the MF.

The WWW scripting scheme (Figure 3) is an example of the minimal integration.

5.2 Basic Integration

This approach is illustrated in Figure 4. The basic idea of this integration approach is to maximally use the existing MF functions (MFFs) to implement the basic SFFs. In order to achieve this, first, delegation managers and agents are modeled as MF managers and agents, respectively. Then the MF is used to realize the basic SFFs:

- Scripts and their execution results are modeled as management information using MF's MIM, and are stored in the MF's MIB.
- Since the scripts and their results are put in the MIB, they can be transfered using the MF management information transporting service. For instance, SNMP or CMIP can be used for the transfer of the scripts and results.
- Script executions are very much like the management resources, whose states are changing, and who can generate events, etc, and therefore is natural to be modeled as managed objects. Consequently, the monitoring and control of the scripting executions can be done using the MF mechanisms.

As with the minimal integration, the SL is still minimally integrated with the MF MIM. However, unlike the minimal integration, the delegation agent acts as a MF intemediate manager rather than a gateway between two frameworks.

The AMO scripting framework falls into this integration category.

5.3 Scripting Language Integration

This integration gives greatest power in all levels of integration. The idea has largely been laid out in the last section. We list a few noteworthy observations in this section:

- Event processing: Since event processing is very important for network management, and scripts may generate and/or process events, SL may need to provide means to achieve these goals. Example:
  - Event specification: Event combination / event hierarchy / event model / temporal model: these may need to be incorporated into the SL.
  - Both SF and MF events may be passed around as a variable and processed as such.
  - Scripts may be registered with events to process them.
  - Scripts may generate events.
- Error handling: Errors occurred in scripts may be controlled using SL error handling constructs.
- Loop control: management loop constructs may be provided by the SL so that loops can be better controlled. This may be useful to control malfunctioned scripts or scripts which take too much computing or network resources such as CPU cycle or bandwidth.
- SL's data types can be divided into two groups:
  - General purpose data types such as Integer, Boolean, Real, Pointer, Array, Record, etc.
  - MIM data types: data types representing MOs, MIB trees, tables, etc.
- How are the MIM data types designed? For example, should there be one data type for all tables, or should there be one data type for each table?
- The idea of tightly integrating the MIM with the SL is nice, but how can it be implemented?
  One issue is whether the SF should cache remote management information, and if the answer is yes, how can it be achieved.
- It is easier to use an OO SL to integrate with the OSI MIM which is OO.

6 Conclusion and Future Work

Based on our decomposition of the management information model, and an analysis of the functional architecture of the general scripting framework, we have explored various ways to integrate the scripting framework and the management framework. We have also identified three levels of integration which are Minimal, Basic, and Language Integrations. From these observations, we have shown that an integrated scripting framework can bring more power and ease of use to network managers and applications.

A direction for future work being currently explored by us is the design of a debugging architecture within the context of an integrated scripting framework. A manager needs mechanisms for debugging and diagnosing errors during script execution; we would like to find out if an integrated scripting framework will assist in the debugger design and how some of the issues raised in this paper will affect the design.

References

[AC96] P. Abbi and S. Ceri. CMIS++: An Extension to the CMIS Standard for Telecommunication Databases. [DSO96].

[CMRW96] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2) (RFC 1905)*, January 1996.

[Com95] International Organization for Standardization. *ISO/IEC DIS 10164-21, Command Sequencer*, 1995.

[COR95] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.0*, July 1995.

[DSO96] *Proceedings of the 7th IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October 1996.

[Gol95] Germán Goldszmidt. *Distributed Management by Delegation.* PhD thesis, Columbia University, New York, NY, 1995.

[IS990a] International Organization for Standardization. *ISO IS9595, Information Technology - Open Systems Interconnection - Common Management Information Service Element*, January 1990.

[IS990b] International Organization for Standardization. *ISO IS9596, Management Information Protocol Specification - Part 2: Common Management Information Protocol*, January 1990.

[KSSZ97] P. Kalyanasundaram, A. Sethi, C. Sherwin, and D. Zhu. A Spreadsheet-Based Scripting Environment for SNMP. In Lazar et al. [LSS97], pages 752–765.

[LS97] 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).

[LSS97] A. Lazar, R. Saracco, and R. Stadler, editors. *Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management.* IEEE/IFIP, Chapman & Hall, 1997.

[OSI93] International Organization for Standardization. *ISO/IEC 10164-5, Event Report Management Function*, 1993.

[SL95] J. Schonwalder and H. Langendorfer. Tcl Extensions for Network Management Applications. In *Proceedings of the Third Tcl/Tk Workshop*, Toronto, Canada, July 1995.

[TMN92] CCITT. *CCITT Recommendation M.3010 - Principles for a Telecommunications Management Network*, October 1992.

[VPK97] N. Vassila, G. Pavlou, and G. Knight. Active Objects in TMN. In Lazar et al. [LSS97], pages 139–150.

[Wal95] S. Waldbusser. *Remote Network Monitoring Management Information Base (RFC 1757)*, February 1995.

[YGY91] 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.

[YTY96] I. Yoda, H. Tohjo, and T. Yamamura. Interpreter Language-Based TMN Agent Systems. [DSO96].