DSOM98 Submission on March 1997

Active Q Adaptor: A Programmable Management System Integrator for TMN

Yoshiaki Kiriha, Motohiro Suzuki,
Ikuo Yoda*, Kouji Yata*, Shoichiro Nakai

C&C Media Research Laboratories, NEC
4-1-1 Miyazaki, Miyamae-ku, Kawasaki, Kanagawa 216, JAPAN
Tel: +81-44-856-2314 Fax: +81-44-856-2229
{kiriha, motohiro, nakai}@ccm.cl.nec.co.jp

***Optical Network Systems Laboratories, NTT**
1-1 Hikarinooka, Yokosuka, Kanagawa 239, JAPAN
Tel: +81-468-59-3290 Fax: +81-468-55-1284
{ikuo, kouji}@exa.onlab.ntt.co.jp

Abstract

This paper discusses the Active Q Adaptor, which can improve TMN by providing integratibility and programmability. Key concepts in this proposed adaptor include a dynamic MO and a message routing function. The dynamic MO, whose attributes are defined in terms of meta-data and whose behavior is implemented with any script languages, is programmable in run time. With the message routing function, which allows management-protocol-independent access of MO, managers are able to efficiently integrate agents that employ diverse managed information models and protocols. Furthermore, the adaptor's CORBA-based distribution of functions makes it scaleable and adds to its high performance. With these features, our programmable Active Q Adaptor is able to create an integrated network management system; this can be applied to end-to-end management of, for example, customer premise networks, as such IP networks linked through ATM backbones.

1. Introduction

Recent deregulation of the telecommunication industry has put increased pressure on service and network providers to develop and deploy new network service and management functions to meet various new customers demands. To succeed in an open and competitive global market, network management functions require three important features: interoperability, integratibility, and programmability [1].

With regard to interoperability, standardization bodies like ITU-T and ISO have already produced a number of useful documents that define information exchange module services and protocols for network operation, administration, and management. The Telecommunication Management Network (TMN) [2], for example, is an ITU-T framework that employs a manager and agent based system architecture and contains Guidelines for the Definition of Managed Objects (GDMO) and Common Management Information Protocol (CMIP), and that has recently gained wide usage in Synchronous Digital Hierarchy (SDH) and Asynchronous Transfer Mode (ATM) broadband backbone networks. By way of contrast, in the Internet community, Simple Network Management Protocol (SNMP) and Internet Management Information Base (MIB) are commonly utilized. Both of these protocol-based approaches, however, lack integratibility and programmability because of their policy of centralized network management and because of their static management of information.

Integratibility has become essential with the recent emergence of newly inter-working domains that employ protocols like CMIP and SNMP and that contain a variety of managed information models. In order to create a network management system (NMS) like that illustrated in **Fig. 1**, for example, with Internet Protocols (IP) networks linked through an ATM backbone network, such integratibility is the first requirement, and the Common Object Request Broker Architecture (CORBA) [3] created by the Object Management Group (OMG) would seem to hold promise for this kind of inter-working. Since backbone and IP networks are

generally managed through CMIP and SNMP respectively, these two must first be integrated to be able to manage end-to-end connections, and for this purpose, the Network Management Forum (NMF) has already published specifications for inter-working CORBA with both CMIP and SNMP [4]. The problem is, however, that the information models and protocols upon which this approach is based are static, which means that it is still impossible to integrate the various proprietary protocols contained in such a network, and that it will be impossible to cope with new protocols that can be expected to be deployed in the future.

The need for greater programmability may be considered from two viewpoints: that of the user and that of network management. In future telecommunication systems, networks first of all ought to be active in the sense that customers ought to be able to reprogram network behavior as it affects them individually [5]. Similarly, from the management point of view, operators ought to be able to reprogram management functions in response to individual needs. For example, in order to reduce the traffic needed to perform management functions and to achieve more efficient management operations, they ought to be able to create newly defined, newly behaving Managed Objects (MOs) during management system executions, but such is not yet possible in any actual network management systems reported to date.

For these reasons, we believe that a new management framework is required that can improve TMN by providing integratibility and programmability. In TMN, Q Adaptor (QA) is defined as containing one or more Q Adaptor Function (QAF) blocks whose responsibility is to interface conversion between the TMN and such non-TMN entities as CORBA, SNMP, and proprietary protocols [6]-[7]. We propose here an Active Q Adaptor (AQA), which we have produced from standard QA by adding programmability and CORBA integrativility.

This paper is organized as follows: Section 2 introduces the concept of AQA and how to accomplish run-time redefinition, and discusses how to achieve the integration of agents that

employ diverse managed information models and protocols; Section 3 discusses the AQA configuration, which consists of distributed function modules, and shows examples of these module interactions; Section 4 describes implementation techniques for improving AQA reliability; and finally, Section 5 presents some concluding remarks.

2. Active Q Adaptor Concept

A logical view of the proposed AQA is illustrated in **Fig. 2**. AQA is located in the middle, between managers and agents. We have chosen to add programmability to AQA, rather than to the agents, because this avoids the need for any modification of MO definitions or behaviors in the agents. In addition to its programmability, AQA also provides managers with a unified view of all information from the agents connected to it.

In our proposed system, *"Activeness"* means the capacity to create newly defined MOs, to modify MO definitions, and to reprogram MO behavior during a system run. Conventional MOs implemented with C/C++ must be recompiled in order for their definitions and behavior to be modified. To avoid this drawback, we have created dynamic MOs that can be implemented with a script language (i.e., Java, Tcl, etc.) As depicted in **Fig. 3**, the dynamic MO consists of three components: a common frame, a meta-data, and a behavior program section. The common frame provides such attribute values as naming, relationship, and pointer value (with respect to meta-data), and is implemented with C/C++. The meta-data is text information that specifies attribute labels and IDs that correspond to individual behavior programs. Behavior programs can be written in any scripting language and provide the accessing and processing of managed information. In dynamic MOs, defined attributes have no value in themselves; they only provide access to behavior programs.

An active MOs introduced in [8] appears to have arisen from similar motivations, but those active MOs are defined as a subclass of an Event Forwarding Discriminator (EFD) MO;

our proposed dynamic MO suffers no such limitations with respect to class derivation. This makes AQA capable of providing a more programmable environment than can the active MO framework proposed in [8].

With regard to creating or redefining dynamic MOs, as well as to adding behavior programs to them or changing existing behaviors, CMIS operations such as M-CREATE and M-ACTION can be utilized as described in [9] because dynamic MO definitions are based on GDMO. Moreover, since the AQA is constructed of CORBA-based distributed modules (which will be discussed in **Sec. 3**), it can accept any CORBA-based requests whose semantics is compatible with CMIP requests. Additionally, if Java is utilized as the script language for behavior programs, other IP based protocols, such as Hyper Text Transfer Protocol (HTTP) and Remote Method Invocation (RMI), can also be used, through only to add or change behavior programs.

In addition to dynamic MOs, the AQA also employs two other types of MOs: fixed MOs and shadow MOs (In order to avoid confusion of these three types of MOs with those MOs that are stored in agent systems, we hereafter refer to this last type as remote MOs.). Fixed MOs and remote MOs are conventional MOs, and their definitions and behavior programs can not be changed. Rather than providing programmability, these fixed and remote MOs provide high performance access functions. Shadow MOs are used to represent individual remote MOs, and they are assigned the naming and relationship attributes of the individual remote MOs to which they respectively correspond. Shadow MOs are used in AQA to create a Management Information Tree (MIT) (see Fig. 4). One of these shadow MOs, is used to represent the root object contained in an agent MIT, and when AQA receives a request to modify the definition or behavior program of a remote MO, 1) AQA creates a new dynamic MO to correspond to the modified remote MO, and 2) it also creates a new shadow MO in MIT between the shadow MO for the root object and the newly created dynamic MO. Such new shadow MOs are used to

represent any remote MO that lies between the root object and the remote MO that is to be modified. We should note that AQA can create newly defined dynamic MOs (i.e., dynamic MO #1 in Fig. 4) for improving the AQA management capability, as well as it creates dynamic MOs (i.e., dynamic MO #2 and #3 in Fig. 4) for modifying remote MOs.

Let us next consider how AQA can use its MIT to execute management operations. When a requested operation only involves either a single dynamic MO or a single fixed MO (i.e., no scope parameters are specified), AQA executes the relevant behavior program and notifies the requesting manager of the execution results. We should note here that behavior programs of dynamic MOs, in their execution, send agents new requests for accessing remote MOs. When the requested operation only involves either a single shadow MO or a single remote MO, AQA forwards the request directly to the appropriate agent. AQA is able to identify the appropriate agent because it knows where all these MOs are located. Hereafter, we refer to all such activities as *"message routing"*. As illustrated in **Fig. 5**, to the degree that AQA simply forwards requests for operations involving a single shadow MO and a single remote MOs, this design serves to keep AQA overhead low.

The process becomes only slightly more complex when requests do specify scope parameters. AQA first translates all scope parameters into a) a set of single dynamic MO operations and b) a request, containing new scope parameters, for a remote MO operation. If the specified scope does not include any dynamic MOs (i.e., includes only remote MOs), AQA forwards the request to the appropriate agents with no modification. If the specified scope includes one or more dynamic MO, AQA modifies the scope parameters by adding filter conditions that eliminate duplicate operations being performed on remote MOs that have already been performed on the dynamic MOs to which they correspond.

The message routing function provides managers with a remote MO access capability that is independent of managed information models and management protocols. This enhances their ability to integrate agents. Additionally, the usage of the dynamic MOs makes it possible for managers to reduce the complexity of the applications that they employ because behavior programs are able to access remote MOs regardless of the varying management protocols utilized by those MOs. In order for AQA to convert various types of agents' managed information models and management protocols into its own type, each conversion module is required to be pluggable in AQA depending on the agents to be managed. To realize such conversion modules, we can utilize both CORBA distributed processing techniques and conversion algorithms. A distributed configuration of AQA is deeply discussed in next section. With regard to the conversion algorithms, NMF has produced useful specifications. For instance, ISO-Internet Management and Control (IIMC) specification [10] deals with the CMIP/SNMP conversion, and Joint Inter-Domain Management (JIDM) specification [11] deals with the CMIP/CORBA conversion.

We have introduced AQA concepts which include a dynamic MO, run-time redefinition of the dynamic MO, and a message routing function. Since AQA can receive any request from both CMIP and CORBA managers and can access agents through any one of CMIP, SNMP, CORBA, or proprietary protocols, it is able to integrate the agents of various networks to create an integrated network management system. We note that the proposed AQA concept enables managers to achieve the following flexible management services;

- Creation of newly defined MOs (i.e., dynamic MOs),
- Modification of definitions and behavior programs on remote MOs,
- Rapid development of integrated management functions, and

- On-the-fly deployment of new management functions.

3. Facilities of Active Q Adaptor Modules

We have designed actual software modules to realize AQA concept described in the previous section. In general, distributed computing techniques are useful for developing a scaleable system with high performance. To implement AQA in a distributed manner, we have categorized the AQA functions into the following four facilities: a management protocol service, a message routing service, a managed information access service, and a meta-data service. Interfaces among distributed modules are designed by using a Interface Definition Language (IDL) [12] taking the language neutral feature of CORBA into account.

Figure. 6 shows a distributed configuration of AQA in which each module corresponds to the four facilities described above. The following describes each module in more detail.

Management Protocol Proxy (MPP)

This module (MPP) exists for every management protocol such as CMIP or SNMP. MPP is responsible for converting parameters specified in management protocol dependent interface to that specified in the IDL interface of a message router which is described next, and vice versa. As described in **Sec. 2**, the specifications from IIMC and JIDM are examples for this converting algorithms. By configuring the management protocol services as separated modules, we can easily add and remove these modules depending what kind of management protocols are employed in networks to be managed.

Message Router (MR)

This module (MR) is a core part of AQA and is responsible for the message routing service. Because of this, MR maintains MIT that consists of dynamic MOs, fixed MOs, and shadow MOs. MR provides two types of interfaces; an agent interface and object interfaces. The agent interface exists for accessing multiple MOs by specifying scope parameters. On the

other hand, the object interfaces exist for accessing individual MOs. We note the definitions of all object interfaces are the same, and do not depend on the definitions of MOs in contrast to the JIDM specification. This characteristic contributes to the performance improvement and to realization of run-time redefinition.

Since all operations go through this module, the processing load is higher than that of the other modules. This is the reason why multithread programming in C/C++ is utilized for the MR implementation. Also, a technique called loaders [13] in Orbix (one of CORBA products) is useful for an efficient execution and memory usage, since it enables the object interfaces to be activated only if they are requested.

Managed Information Access Server (MIAS)

This module (MIAS) stores behavior programs of dynamic MOs, and it is responsible for invoking an appropriate behavior program according to a request from MR. The interface for invoking behavior programs consists of CMIS compatible methods such as a get, set, create, delete, action, and notification. Each method is required to specify a class name of a dynamic MO on which requested operation is performed. Since this interface creates only one CORBA object, AQA saves memory consumption in spite of using CORBA technology.

Although MIAS can be implemented any one of script languages, Java seems to be a best solution for implementing this service. This is because that IDL to Java mapping [14] has already standardized by OMG. If a large amount of dynamic MOs are employed and a large number of operations are requested, it is useful to introduce mirrored MIASs in order to efficiently execute behavior programs. The CORBA distributed processing technology is easily able to achieve this duplication.

Meta-Data Server (MDS)

This module (MDS) stores a meta-data which consists of the definitions of MO class, attributes, notifications, and actions. The structure of stored information is compatible with the

Management Knowledge Management Function (MKMF) [15] specification by ISO. Then the MDS provides the other modules with an interface for accessing the meta-data. Furthermore, it coordinates the creation and modification of dynamic MOs by controlling MIAS and MR.

In the rest of this section, we show examples of AQA behavior concerning a dynamic MO creation and a dynamic and remote MOs access, as illustrated in **Fig. 7** (a) and (b) respectively. We describe the dynamic MO creation at first. The creation request is informed to MDS through MPP and MR (1-3). MDS requests MIAS to install a new behavior of the specified dynamic MO (4-5), then requests MR to create the dynamic MO and associating shadow MOs (6-7) in MIT. Finally, MDS replies the operation result to the manager through MPP and MR (8-10). This dynamic MO creation enables managers to modify definitions of remote MOs and to move management intelligence into AQA. For instance, this is useful when MO definitions in multi-vendor agents are different due to different interpretations of ambiguous text specification and selections of optional specifications.

With regards to the dynamic and remote MOs access in Fig.7 (b), we assume that a manger sends a request with scope parameters. The request is received by MPP (1), and is forwarded to MR (2). MR converts the request into a set of single dynamic MO operations (3-10) and a request for a remote MO operation (11-14). Concerning the single dynamic MO operation, MR invokes MIAS to execute an appropriate behavior program (3). The behavior program in MIAS accesses agents in order to operate remote MOs through MR and MPP (4-9). The results from the agents is processed in MIAS, then the results of the dynamic MO access return to MR (10). After MR receives the access result from dynamic MOs (10) and remote MOs (14), it replies the final operation results to MPP and the manager (15-16). This dynamic and remote MOs access enables AQA to provide managers simple APIs for invoking sophisticated and rich functions which achieve an integrated network management. For instance,

this is able to realize an intelligent alarm filtering function which is adaptable to an alarm importance and a degree of management network congestion, and a complex connection setup function which consists of such multiple remote MO access as Connection Termination Point (CTP) MO creations, Trail Termination Point (TTP) MO creations, and cross connect MO actions.

4. Implementation Techniques

The main facilities for providing integratibility and programmability with AQA modules are described in Sec.3, however, AQA should also take the following issues into account for improving the system reliability. One is a version control of behavior programs in MIAS, and another is a transaction management in MR.

4.1 Version Control of Behavior Program

Since MR can invoke many requests by using multi-thread execution, both execution and modification for the same behavior program can be requested concurrently. Even if such situations occur, MIAS should ensure the consistent execution of behavior programs.

Java, one popular script language, can provide a useful mechanism for dealing with the above problem. This mechanism is called a class loader [16], which loads necessary Java classes from files to a main memory and organizes one naming space for those classes. Since behavior programs correspond to Java classes in MIAS, the behavior program modification can be achieved by overloading the corresponding Java class.

In order for MIAS to ensure the consistent execution of behavior programs, it invokes multiple class loaders and instantiates different versions of behavior programs in different naming spaces. As results of these, the behavior program execution in one naming space can be processed independent of the behavior program modification in another naming space. When all behavior program executions are completed, then the old version of behavior program is

discarded automatically. We should note here that MIAS manages a memory cache for loaded behavior programs and extends the default class loader to check the cache taking the performance improvement into account.

4.2 Transaction Management

Since there is no limitation for describing behavior programs, a dynamic MO can invoke other dynamic MOs access as well as multiple remote MOs access. In order to ensure the atomicity of requested operations in such a situation, the MR is required to support a transaction management capability. We utilize a two-phase commitment protocol in which operations are executed according to the prepare and commit/abort phases.

As a result of introducing the two-phase commitment protocol, each behavior program needs to specify procedures corresponding to the prepare, commit, and abort phases. Then MR controls the behavior program execution by indicating the current phase. Furthermore, MR and MIAS is necessary to share a transaction identifier as one of the interfacing parameters. By using this, MR avoids a conflict access to a dynamic MO which has already been involved in another transaction.

Any of the existing management protocols do not support transactional operations. Because of this, behavior programs for a prepare phase must store old values of operating MO attributes. In case that some failure is detected, behavior programs for an abort phase must invoke new operations which restores the operated attribute values to old values in order to rollback the performed operations in this transaction.

5. Conclusion

The proposed AQA is a programmable integrator for the TMN compatible and non-TMN management systems. A dynamic MO which utilizes script languages enables managers to customize their management applications by using CMIS or CORBA-based requests at run-

time. A message routing function provides an efficient integration capability of agents which employ multiple managed information models and protocols. A distributed AQA design applying CORBA technology makes AQA scaleable with high performance. Furthermore, a version control mechanism of behavior programs and a transaction management mechanism improve the AQA reliability.

We believe that AQA is able to realize end-to-end management for heterogeneous telecommunication networks. Based on the proposed design, we have been implementing a prototype system for validating the feasibility and applicability of our concept into actual ATM backbone networks with IP functionality.

References

[1] A. Manley, C. Thomas. **Evolution of TMN Network Object Models for Broadband Management**. IEEE Communications Magazine, 35(10):pp60-65, 1997.

[2] ITU-T Recommendation M.3010, **Principles for a Telecommunications Management Network**, 1992.

[3] OMG. **The Common Object Request Broker Architecture and Specification**. 1995.

[4] N. Soukouti, U. Hollberg. **Joint Inter Domain Management: CORBA, CMIP and SNMP**. In Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network Management, San Diego, USA, May 1997, pp153-1164.

[5] D. Tennenhouse, D. Wetherall. **Towards an Active Network Architecture**. ACM Computer Communication Review, 26(2):pp5-18, 1996

[6] R. Glitho, S. Hayes. **Approaches for Introducing TMN in Legacy Networks: A Critical Look**. IEEE Communications Magazine, 34(9):pp55-60, 1996.

[7] A. Keller. **Tool-based Implementation of a Q-Adaptor Functions for the seamless Integration of SNMP-managed Devices in TMN**. In proceedings of the IEEE/IFIP 1998 Network Operations and Management Symposium, New Orleans, USA, February 1998, pp400-411.

[8] A. Vassila, G. Pavlou, G. Knight. **Active Objects in TMN**. In Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network Management, San Diego, USA, May 1997, pp139-150.

[9] T. Goto, H. Tohjo, I. Yoda. **GDMO and Behavior Program Transmission in TMN Agent**. In proceedings of the 8th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, Sydney, Australia, October 1997, pp156-166.

[10] NMF. **Translation of Internet MIB II to ISO/CCITT GDMO MIB**. December 1995

[11] NMF. **Inter-domain Management: Specification Translation**. Open Group Preliminary Specification, March 1997.

[12] OMG. **OMG IDL from CORBA v2.1**. August 1997.

[13] S. Baker. **CORBA Distributed Object; Chapter 20 Loaders: support for persistent objects**. ACM Press (ISBN 0201-92475-7), 1997, pp329-343.

[14] OMG. **IDL/Java Language Mapping**. OMG TC Document orbos/97-03-01.

[15] ISO/IEC 10164-16. **Managed Knowledge Management Function**. 1995

[16] J. Gosling, H. McGilton. **The Java Language Environment: A White Paper**. Sun Microsystems, May 1996..

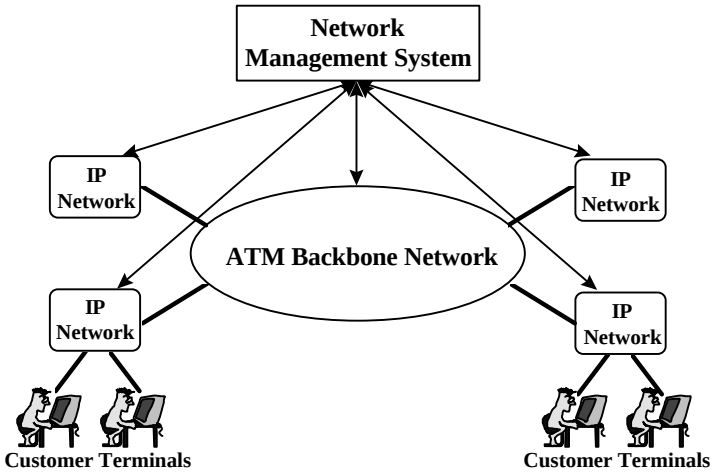


Figure 1. End-to-End Management Example

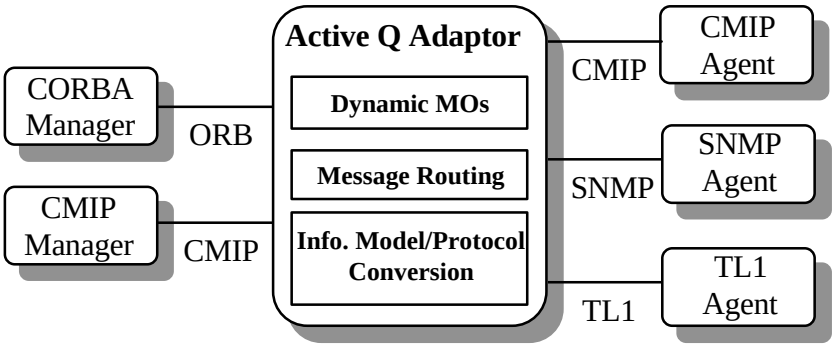


Figure 2. Logical View of Active Q Adaptor

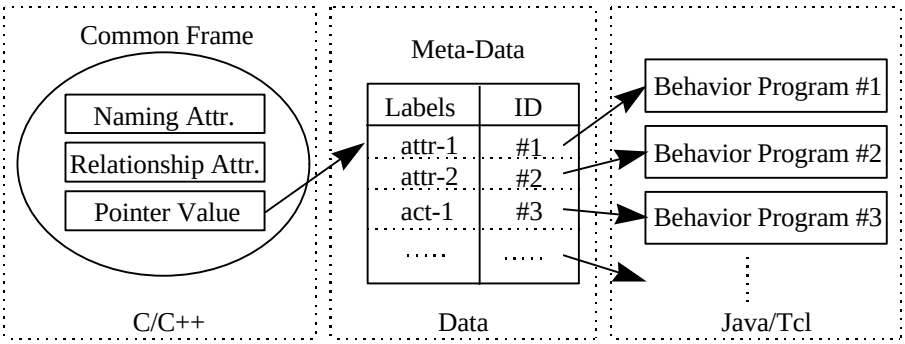


Figure 3. Dynamic MO

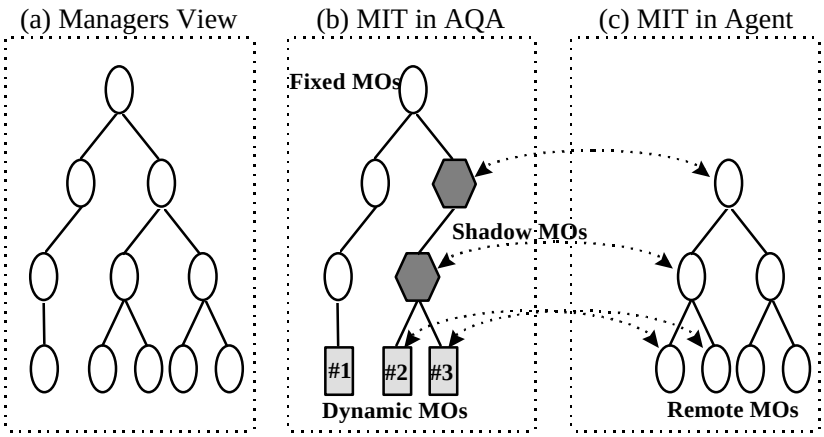


Figure 4. MITs in Manager, AQA, and Agent

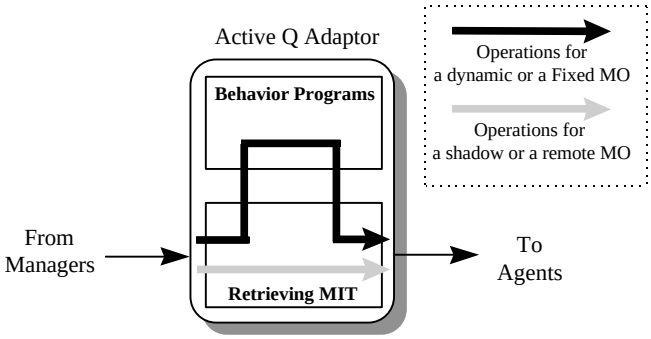


Figure 5. Message Routing Capability

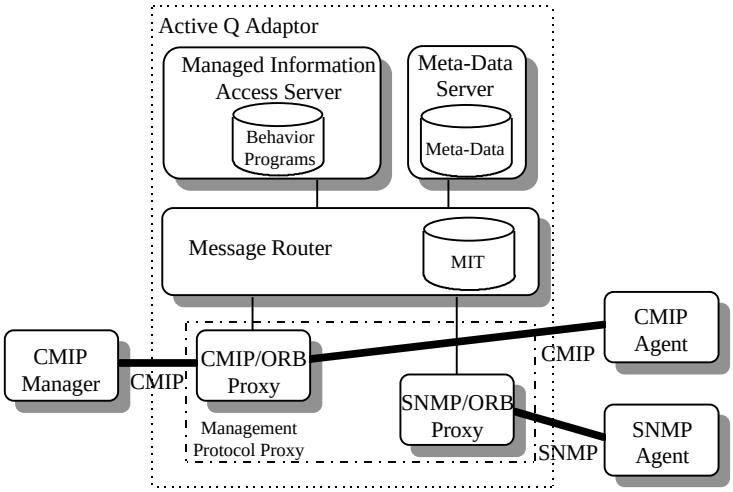
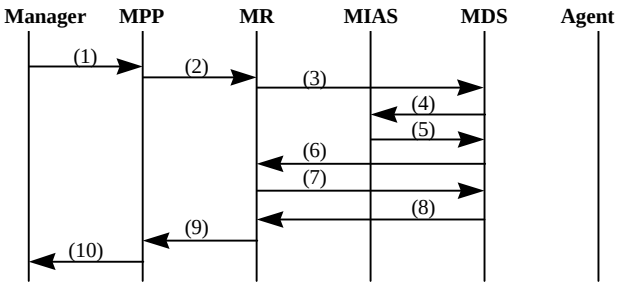
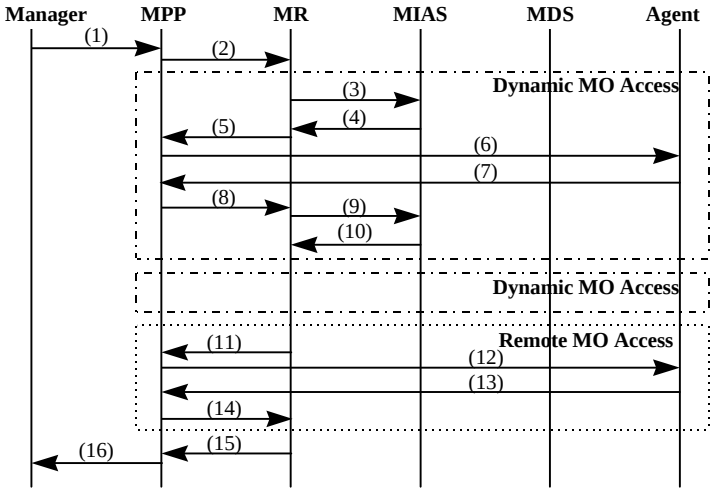


Figure 6. Distributed Design of Active Q Adaptor



(a) Dynamic MO Creation



(b) Dynamic and Remote MOs Access

Figure 7. Message Sequence Examples in AQA