MOBILE AGENT MANAGEMENT

By

Patricia Cuesta Rivalta

A Thesis Submitted to

the Faculty of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Master of Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada

October 2000

© Copyright Patricia Cuesta Rivalta, 2000

The undersigned hereby recommended to

Faculty of Graduate Studies and Research

acceptance of this thesis

MOBILE AGENT MANAGEMENT

Submitted by Patricia Cuesta Rivalta, B.Sc.

in partial fulfillment of the requirements

for the degree of Master of Engineering

Department Chair

Thesis Supervisor, Professor Bernard Pagurek

Carleton University

October, 2000

ABSTRACT

The development of network management systems based on decentralized paradigms have been proposed to address the problems that arise because of the rapid growth, the active and dynamic nature, heterogeneous environment, and geographical and administrative distribution today's telecommunication networks face.

Mobile agents are considered an essential technology in the development of distributed software applications because of their capabilities to move across distributed environments, integrate with local resources and other mobile agents, and communicate their results back to the user or authority on behalf they are acting. Particularly in network management systems, the decentralization of management functionality benefits from their integration with the mobile agent technology.

This thesis extends the *Mobile Code Toolkit* of the **Perpetuum Mobile Procura** project for integration with an SNMP agent. Mobile agents are provided with management capabilities to dynamically access and extend the *XMS-SNMP agent*'s MIB. Based on this architecture, it designs and implements a mobile agent MIB allowing manager application to monitor and control the mobile agents visiting the network.

ACKNOWLEDGEMENTS

I would like to thanks to my thesis supervisor Professor Bernard Pagurek for his invaluable guidance, support and encouragement during these two years, thanks for give me the opportunity to share ideas and experiences with the colleagues at the Network Management Laboratory and for his decisive recommendation that led me to join the Canadian telecommunication industry.

Special thanks to my wonderful friends who have always been there when I needed.

Most importantly, thanks to my family- *papi*, *mami*, and *Lalita* - for their encouragement and support for many years to get here and continue. I owe them so much.

Thanks.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
AP	Agent Platform
API	Application Programming Interface
ASN.1	Abstract Syntax Notation
ATP	Aglet Transport Protocol
BER	Basic Encoding Rules
CF	Communication Facilitator
CIM	Common Information Model
CMU-SNMP	Carnegie Mellon University SNMP agent
CORBA	Common Object Request Broker Architecture
CORBA-IIOP	CORBA Internet I O Protocol
CS	Communication Service
DAE	Distributed Agent Environment
DF	Directory Facilitator
DISMAN	DIStributed MANagement
DPI	Distributed Protocol Interface
EMANATE	Enhanced MANagement Agent Through Extensions
FIPA	Foundation for Intelligent Physical Agents
IA	Intelligent Agent

IDL	Interface Definition Language
IETF	Internet Engineering Task Force
IKV++	Innovation+Know-How+Vision
ITU-T	International Telecommunication Union- Telecommunication Standard Sector
JASMIN	Java Script MIB ImplementatioN
JIDM	Joint Inter Domain Management
JDK	Java Develoment Kit
JMX	Java Mangement eXtensions
KQML	Knowledge Query and Manipulation Language
M2M-MIB	Manager-to-Manager MIB
MA	Mobile Agent
MAF	MIB Access Facilitator
MASIF	Mobile Agent System Interoperability Facility
MbD	Management by Delegation
MBean	Management Bean
MCD	Mobile Code Daemon
MCE	Mobile Code Environment
MCM	Mobile Code Manager
MCMIB	Mobile Code MIB
MCMIBTree	Mobile Code MIB Tree
MCT	Mobile Code Toolkit
MF	Migration Facilitator

MEF	MIB Extender Facilitator
MHMIB	Migration History MIB
MIB	Management Information Base
MLM-MIB	Mid-Level-Manager MIB
NE	Network Element
NMS	Network Management System (Station)
OMG	Object Management Group
OSI-SM	Open System Interconnection System Management
PMP	Perpetuum Mobile Procura project
RDF	Resource Description Framework
RDP	Remote Delegation Protocol
RDPI	Reverse DPI
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
TNM	Telecommunication Network Management
UCD-SNMP agent	University of California at Davis SNMP agent
VMC	Virtual Managed Component
WAN	Wide Area Network
LAN	Local Area Network
XMS-SNMP agent	eXtensible Mobile Supported SNMP agent

Chapter 1 Introduction

1.1 Problem Overview

The development of network management systems based on decentralized paradigms have been proposed to address the problems that arise because of the rapid growth, the active and dynamic nature, heterogeneous environment, and geographical and administrative distribution today's telecommunication networks face. The Management by Delegation (MbD) model [62] has provided a management architecture based on a distribution paradigm in such a way that management functions can be *dynamically* distributed among management entities in a network management system. At runtime, managers *delegate* management tasks to MbD *agents*. The IETF DIStributed MANagement (DISMAN) Working Group [17] has proposed a distributed model for the Internet community based on the SNMP protocol where the management functionality is distributed among multiple network managers: the *Schedule* and the *Script* MIBs provide mechanisms for performing management operations distributed temporally and spatially.

Dynamic extensibility is one of the solutions that has been used to support the MbD model [13] in network management environments: *extensible agents* are MbD agents that model a distributed MIB in a master/subagents architecture where the MIB consists of a *static* MIB residing in the master agent and several distributed subagents that *dynamically* register portions of the MIB.

More recently, *Sun Microsystems Inc*. has also led the development of a set of standard specifications to equip the Java technology with a universal extension for distributed management, known as the Java Management eXtension (*JMX*) [52].

On the other hand, mobile agents are considered an essential technology in the development of distributed software applications because of their capabilities to move across distributed environments, integrate with local resources and other mobile agents, and communicate their results back to the user or authority on behalf they are acting. Particularly in network management systems, the decentralization of management functionality benefits from their integration with the mobile agent technology.

The Object Management Group (OMG) [35] and the Foundation for Intelligent Physical Agents (FIPA) [11] are the two major standardization bodies related to agent technology that are addressing interoperability of agents from different manufacturers in order to fulfill the requirements of today's dynamic, heterogeneous and distributed service provisioning applications.

This thesis is part of the research work of the **Perpetuum Mobile Procura** project [40, 6] devoted to advanced network management models using mobile agent technology. First, the network management model is based on the SNMP protocol, the Internet framework for managing heterogeneous networks. Second, the provision of extensibility capabilities to legacy SNMP agents using the Distributed

Protocol Interface and the Reverse Distributed Protocol Interface protocols, and third, the use of mobile agents to dynamically augment the local SNMP agent functionality. Mobile agents do not replace the SNMP protocol used by classic network management applications to interface with the SNMP agent services in the network elements; instead, they complement the management application – SNMP agent interaction model providing more efficient solutions for managing today's heterogeneous, dynamic, and complex networks.

Exploiting their migration capabilities, mobile agents arrive to the same network element of and interact with the SNMP agent taking advantages of the local communication and the management capabilities the resident SNMP agent already has. Two main types of mobile agent – SNMP agent interaction are identified:

1. Interaction derived from arriving mobile agents carrying with management information and wanting to extend the local SNMP MIB. A special purpose protocol need to be used to handle the two-way communication: the registration/deregistration processes of MIB variables by mobile agents with the resident SNMP agent, and the request/response of MIB variables forwarded by the SNMP agent to the mobile agent actually hosting the MIB sub-tree. Obviously, the SNMP agent must be modified to support the dynamic MIB multi-registration and to schedule queries for the distributed MIB variables; and mobile agents would employ an API of a local service to accomplish the registration and be able to accept/reply to queries to the hosted MIB variables.

2. Interaction derived from arriving mobile agents requesting data from the SNMP MIB can be governed by the provision of either the SNMP protocol or a lightweight and more efficient version of the SNMP protocol with the complete managerial capabilities of the SNMP. Employing the SNMP protocol, the SNMP agent does not have to be modified but mobile agents have to have full SNMP managerial capabilities and have to handle tasks as BER encoding between two co-resident processes. The other alternative proposes a lightweight protocol that provides mobile agents with access to the SNMP MIB variables without having to undertake the BER encoding and decoding as SNMP has to.

The thesis focuses in the integration of a mobile agent infrastructure with an extensible SNMP-based network management system. The *XMS-SNMP master agent* uses the lightweight SNMP Distributed Protocol Interface (DPI) as the protocol supporting communication between the master agent and subagents that, taking advantage of the master agent and subagents locality, do not deal with ASN.1 details and BER encoding rules as the SNMP protocol does. The lightweight SNMP Reversed DPI (RDPI) protocol is used to accept management requests from local manager applications wishing to access MIB variables.

The thesis implements an architecture [57, 37] to extend the *Mobile Code Toolkit* framework for integration with the extensible *XMS-SNMP agent*. The solution provides mobile agents carrying with management information can extend the

SNMP agent's MIB using the DPI protocol [57, 37], and mobile agents with access capabilities to retrieve information from the SNMP agent's MIB using the RDPI protocol.

Based on this architecture, the convenience of using RDPI instead of the SNMP protocol in a local mobile manager application - SNMP agent interaction needs to be measured .

This thesis also addresses the problem of managing mobile agents. For the above management infrastructure to be feasible and complete in providing the needed services to network managers, mobile agent management supports should be provided. It designs and implements a mobile agent MIB for manager applications to monitor and control the mobile agents visiting a *Mobile Code Toolkit* - *XMS-SNMP agent* platform.

1.2 Thesis Contributions

This thesis has two major contributions. The first contribution is the implementation of the dual component architecture to extend the *Mobile Code Toolkit* providing mobile agents with SNMP management services to dynamically access and extend the *XMS-SNMP agent*'s MIB. One component provides MIB access capabilities to mobile agents wishing to retrieve MIB information from the extensible SNMP agent. It offers a standard interface for mobile agents to submit requests to the *XMS-SNMP agent*. The second component encapsulates the MIB extension capabilities for mobile agents wishing to act as subagents extending the SNMP agent's MIB. It offers mobile agents the functions of register and unregister the management information they provide with the *XMS-SNMP agent*, applies management control, and forwards the requests from the *XMS-SNMP agent* to these MIB extenders.

The second contribution is the design and implementation of a Management Information Base (MIB) for managing mobile agents running in a *Mobile Code Toolkit* - *XMS-SNMP agent* integrated environment. The management MIB has been defined to monitor and control the number and the attributes of mobile agents visiting the mobile agent system. The mobile agent MIB provides information of the mobile agent identity, class, type, current location, execution status, migration and communication capabilities and their migration history. It also provides management functions to *start*, *suspend*, *resume*, *stop* and *destroy* mobile agents. The MIB is actually implemented in a stationary agent that interfaces mobile agents visiting the mobile agent system and registers the MIB variables with the local *XMS-SNMP agent*.

1.3 Thesis Outline

In the remainder of the thesis, Chapter 2 examines some of the standardization activities being done in the development of network management systems based on decentralized paradigms and presents the functional architecture of the *XMS-SNMP agent*, the dynamically extensible SNMP agent being used in the integrated environment devoted to network management in the **Perpetuum Mobile Procura** project. It describes the use of the SNMP Distributed Protocol Interface [57] protocol to allow resident mobile agents, acting in the role of subagents, to dynamically extend the existing SNMP MIB. The Reverse DPI (RDPI) [57, 37] protocol is also used to allow resident mobile agents to access the SNMP MIB saving in both message overhead and message processing compared with the SNMP protocol.

Chapter 3 makes a comparative review of the standardization initiatives for agent systems interoperability, some of their implementations and presents *the Mobile Code Toolkit* as the mobile agent infrastructure of the **Perpetuum Mobile Procura** project. Brief description of its functionality, how mobile agents are managed, the location services and the mobile agent migration history in the *Mobile Code Toolkit* framework are also provided.

Chapter 4 discusses how this thesis provides the *Mobile Code Toolkit* with specialized SNMP management services and presents the dual Virtual Managed Component architecture by the virtue of which mobile agents can integrate with a resident *XMS-SNMP agent*. The *MIBExtendFacilitator* component is the Virtual Managed Component implementation that provides a standard interface to the mobile agents wishing to extend the *XMS-SNMP agent*'s MIB. It actually encapsulates the use of the DPI protocol as the means to communicate with the local *XMS-SNMP agent*. The second, the *MIBAccessFacilitator* component is the Virtual Managed Component that, using the RDPI protocol, provides MIB access capabilities to mobile agents wishing to retrieve information from the extensible SNMP agent's MIB.

Chapter 5 presents the structure and content of the mobile agent MIB and describes its implementation. This project proposes a mobile agent MIB based on the mobile agent management functionality, location services and migration history provided by the *Mobile Code Toolkit* as described in Chapter 3. The MIB is implemented in a stationary subagent, known as the *MCMIBExtender*, that interfaces the mobile agents visiting the network element's mobile agent system and registers the mobile agent MIB variables with the resident *XMS-SNMP agent*'s MIB via the *MIBExtenderFacilitator*.

Chapter 6 focuses on the development of test strategies and cases to demonstrate the efficiency and robustness of the solutions presented in the thesis. It concerns measuring and comparing the response times of RDPI and SNMP management operations. The main result being that RDPI is about three times faster than SNMP's, demonstrated the benefits of using RDPI in a local communication scenario. The mobile agent MIB implementation is also tested and demonstrates how network management applications can manage mobile agents in a Mobile *Code Toolkit* - *XMS-SNMP agent* integrated environment.

Chapter 7 summarizes the research work and it proposes two directions for future work in the area of agent management.

Appendix A shows a concise agent’s execution status transition graph. Appendix B provides the formal managed object definitions for the mobile agent MIB. Appendix C shows an example of the interaction between *XMS-SNMP agent* and *Mobile Code Toolkit* components during an SNMP management request process in a regional mode management scenario. Appendixes D and E shows two test cases of the mobile agent MIB, in local and regional mode management scenarios respectively.

Chapter 2 Distributed Network Management

2.1 Introduction

This chapter examines some of the activities being done in the development of network management systems based on decentralized paradigms: more exactly, the integration of centralized and decentralized paradigms as a solution to the rapid growth, heterogeneous environment, and geographical and administrative distribution today's telecommunication networks face.

Although initially emerged based on a centralized paradigm, the SNMP itself has been continuously evolving into a more effective and distributed (decentralized) network management framework. Let briefly summarize this on-going evolution.

One of the first decentralized approaches appeared as early as the SNMPv1; it is the remote network monitoring (RMON) [56, 50]. The basis of RMON is remote management devices, called monitors or probes, which once configured in remote LAN segments, monitor, compile, and provide summarized statistical information to the network managers.

Another approach seeking to address scalability within the SNMP protocol suite is distributing the network management functions among multiple network managers.

The SNMPv2 MIB module, known as the Manager-to-Manager (M2M) MIB, supported manager-to-manager communications. In other words,

> this MIB provides the means for one management station to request management services from another management station [7].

In general terms, the SNMPv2 protocol operation was extended with the InformRequest/Reponse PDUs sequence to *notify* a manager of management information associated with another manager. Although historic, the M2M functionality remains in SNMPv3 and in the IETF DISMAN Working Group activities. The SNMPv3 framework is viewed as a co-operating set of distributed *SNMP entities*, composed of several interacting modules, which may implement SNMP capabilities for acting either as a traditional SNMP manager, a SNMP agent, proxy, or a combination of both. A network management system based on SNMPv3 can also distribute the management functionality amongst distributed SNMP entities all over the management system. However, because the SNMP is unable to poll a huge amount of management information across low-bandwidth or non-permanent links, this framework does not scale well in very large and distributed networks.

On the other hand, the Management by Delegation (MbD) model provides a management architecture based on a distribution paradigm in such a way that management functions can be *dynamically* distributed among management entities in a network management system. At runtime, managers *delegate* management procedures to MbD *agents*. That is, they move the *code* closer to the managed objects taking advantage of the agent computational power and of the local access to the managed data. This implies not only that the MbD model is an application of *mobile code technology* [41] and that the execution of this code in the agent side is independent or requires minimal intervention of the manager, but also that the MbD model provides management capabilities to these delegated procedures.

For instance, a version of the MbD model based on the SNMP framework was implemented at *SNMP Research Inc.* [43, 22]. This prototype uses the SNMP protocol as the delegation protocol for managers to delegate management procedures, called *scripts,* into remote SNMP agents. Scripts are written in a special purpose programming language, known as SNMP Script Language (SSL). The Mid-Level-Manager MIB (MLM-MIB) was defined for managers for *pushing* scripts into, for *launching* and *controlling* the scripts' execution and for *pulling* the results from the remote SNMP agent. *SNMP Research Inc*.'s prototype inspired part of the work the IETF DISMAN Working Group was charged with.

Another MbD prototype, designed at Columbia University, is based on *elastic processing* [41, 30]. In a network management system scenario based on elastic processing, a *delegator process* (manager application) transfers *delegated procedures* to an *elastic server* (MbD agent) via a *Remote Delegation Protocol* (RDP). An elastic server is an elastic process that exports its dynamically changing interface; that means, upon request, the elastic server supports the extension and contraction of its code and its execution state. The RDP is used by the delegator process to interact remotely with the elastic server. The RDP allows a delegator process to request the incorporation or deletion of a delegated procedure into the elastic server, changing its interface. Delegators can remotely invoke delegated procedures on separate threads in the address space of the elastic process and control its execution as well. MbD agents can be organized as elastic servers and therefore their management functionality can be *dynamically extended*.

Section 2.2 analyses the IETF DIStributed MANagement (DISMAN) Working Group [17] activities being developed to support the MbD model and produce a standard distributed management framework based on the SNMP protocol. Section 2.3 presents other approaches based on the extensibility of management entities participating in a network management system and finally, the functional architecture, the SNMP, DPI and RDPI message scheduling and the multi registration capabilities of the dynamically extensible SNMP agent being used in the integrated environment devoted to network management in the **Perpetuum Mobile Procura** project.

2.2 Distributed Management in the Internet Community

The IETF DISMAN Working Group has taken the original idea behind the *SNMP Research Inc.'s*, to distribute the management functionality amongst multiple network managers. Here are two parts of one of the DISMAN approaches. The Schedule MIB [23] provides a mechanism for performing management operations distributed over time (temporal distribution). The Script MIB [24] provides a mechanism for delegating some kind of executable code (script [22]) for performing management tasks in remote network elements (spatial distribution).

2.2.1 The DISMAN Scheduling MIB

The Scheduling MIB (*schedMIB*) provides the support for network management systems to launch scheduled *SNMP management events* at certain times.

The *schedMIB.schedObjects* subgroup details the scheduled management events; it contains a scalar object, the *schedLocalTime*, providing the local time information used by the *scheduler*[1] and a tabular object, the *schedTable*, where the scheduled events are maintained.

Figure 2.1 shows the *schedMIB.schedObjects* MIB group structure as defined in [23].

schedTable is doubly-indexed by the *schedOwner* and *schedName* attributes allowing multiple users to schedule events on an agent and a user to schedule multiple events.

schedType indicates the type of scheduling the event in a particular entry. There are three types of scheduling:

- Periodic. The management operation is triggered at particular time interval. For instance, a set operation can be scheduled every 24 hours (once a day). The *schedInterval* attribute indicates this time interval.

- Calendar. The management operation is triggered on a weekday-month-day-hour-minute basis. For instance, a set operation can be scheduled on Friday-*-*-23-00 basis (every Friday at 11 pm). *schedWeekDay*, *schedMonth*, *schedDay*, *schedHour*, *schedMinute* attributes provide the scheduling time for a calendar event.

- One-shot. The management operation is scheduled on the same basis as the calendar but the operation itself is triggered once and then disabled.

However, the Scheduling MIB provides capabilities for setting only a *single integer* and *local* MIB variable per event. The *schedVariable* attribute indicates which local MIB variable's object identifier will be written to the value indicated by the *schedValue* attribute. The *set* operation triggers an action that can produce a transition in the state of the managed object. More complex management scenarios wishing to schedule multiple variables and/or non-integer data require the combination of this management MIB with other ones. For example, combining the Schedule and the Script MIBs, management functions can be delegated to remote network devices and launched on a time basis.

[1] A *scheduler* identifies an schedMIB implementer, which is responsible for launching scheduled events.

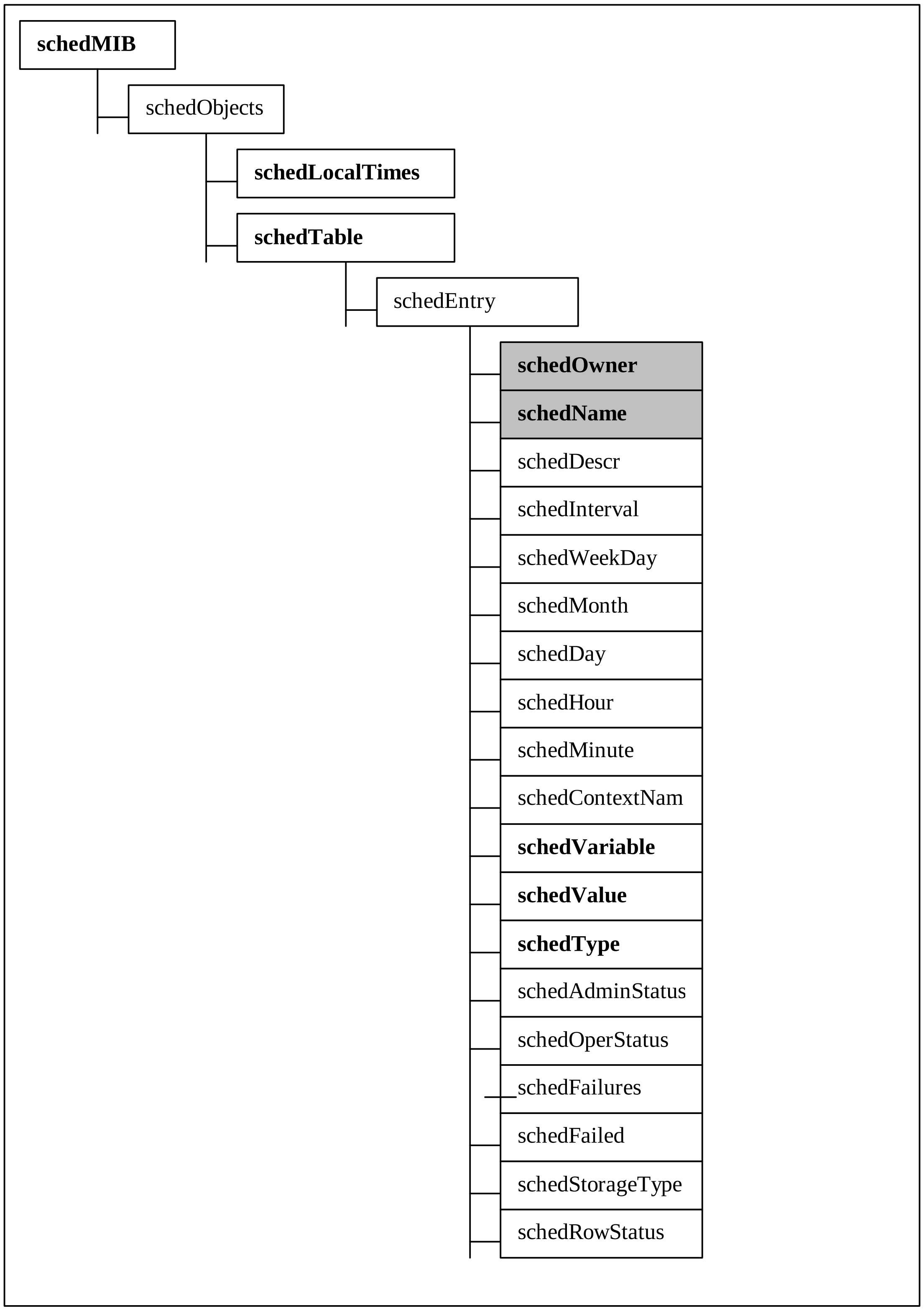


Figure 2.1 The DISMAN Scheduling MIB

The experiences obtained in the implementation of the Scheduling MIB at *SNMP Research Inc*. [26] suggests that an extensible SNMP agent[2] architecture can be used to overcome the capability of the Scheduling MIB to access only local MIB variables. They come up against the drawbacks of having the *schedTable* doubly indexed with the owner and event name attributes when doing ad hoc SNMP get and set requests and the fact that general MIB browser tools do not support octet strings table row indices in a human readable format. They strongly advise about the possibility of the occurrence of the *cumulative processing delay* and the *daylight saving time changes* problems if they are not correctly addressed.

2.2.2 The DISMAN Script

The Script MIB module is an approach to integrate the MbD model in the Internet network management framework. It provides a SNMP based network management system with capabilities for a manager to *delegate* management scripts into SNMP entities acting in the agent role (*distributed managers*).

Distributed managers who implement the Script MIB provide the mechanisms to move the code and to launch the execution of management scripts being delegated by managers and allow managers to control the life cycle of the delegated scripts.

The Script MIB consists of six tables structured as shown in Figure 2.2.

[2] A discussion about extensible SNMP agents comes in section 2.3.

The *smLangTable* and *smExtnTable* provide information about the languages and language extensions supported by the distributed manager.

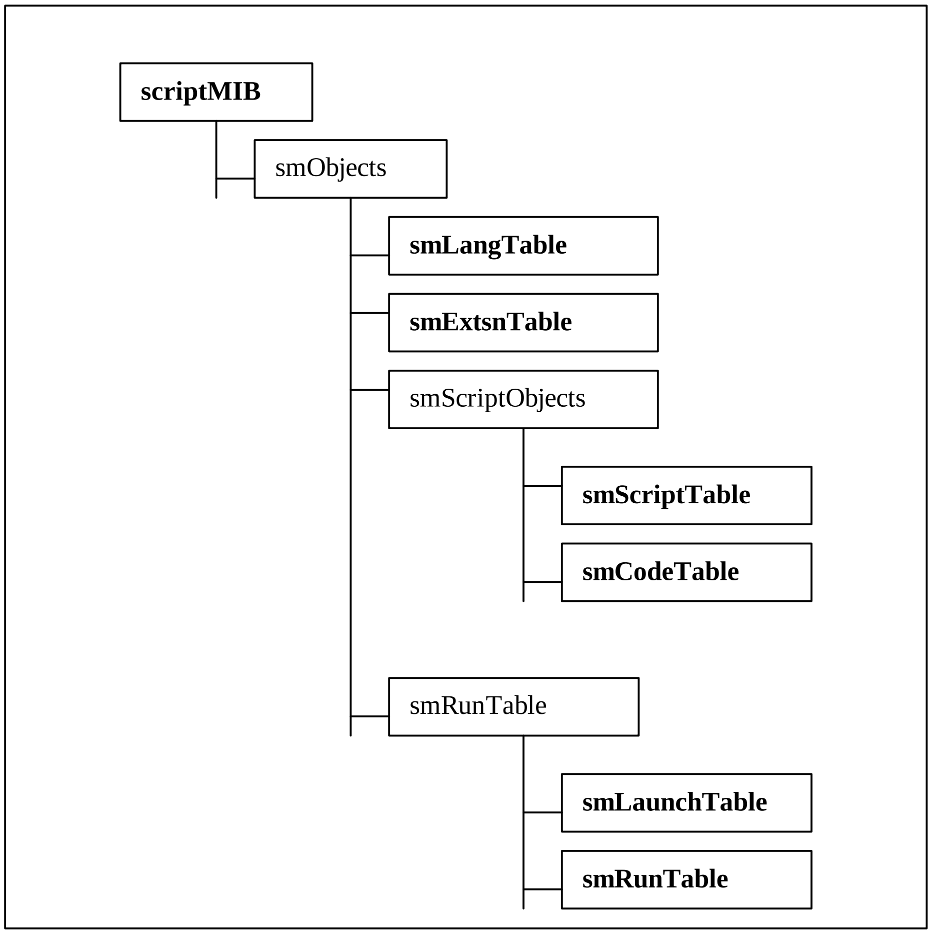


Figure 2.2 The DISMAN Script MIB

The *smScriptTable* maintains a list of all management scripts known to that particular distributed manager. The Script MIB supports two mechanisms for moving script codes to distributed managers, the *push-model* and the *pull-model*. Pushing a script probably requires managers to split the script code into pieces and move them to the distributed manager in a sequence of SNMP *set* requests and store them in the *smCodeTable*. Pulling requires managers to supply the URL specifying the location of the script code and the protocol for retrieving it via a

SNMP *set* request. The URL is written down into the *smScriptTable* and only the Script MIB implementation is responsible for downloading the code from the URL.

The Script MIB provides the mechanism for executing, managing the execution, and retrieving the state and the results of delegated scripts. The *smLaunchTable* and the *smRunTable* describe the scripts that are ready to execute and control its execution respectively. Managers can start, suspend, resume, and stop executing scripts; they can also retrieve the current execution status of scripts, error codes, and their results.

However the delegation of management scripts in the Script MIB is accomplished under the Internet network management framework, the manager-agent interaction exhibits certain degree of granularity, called *micro-management*, through the delegation process. Managers must issue various SNMP *set* requests and poll particular variables until a certain value indicates the termination of asynchronous actions at the distributed manager side.

Managers must also micro-manage the updating of management scripts by stepping the distributed managers through it.

For either uploading or downloading management scripts, the Script MIB does not provide a higher level naming mechanism but the IP addressing scheme. For uploading management scripts, managers use the SNMP protocol and therefore the distributed manager is located via its IP address. For downloading management scripts, distributed managers use the scripts' URL previously written by the manager; again, references are obtained from an IP address.

Nevertheless, although these and other problems that arise because of the low level and fine grained interactions the SNMP protocol imposes for accessing MIB variables, the delegation of management functions the IETF DISMAN framework has proposed, enhance the Internet management framework towards a distributed management environment. It distributes management functionality upon several and collaborative SNMP entities, managers and distributed managers; it is able to cope with situations that arise over non-permanent network connections. The IETF DISMAN Working Group is also working on the Event, Expression, and Notification MIBs to complement the management functionality distribution.

Two experiences in the implementation of the IETF Script MIB also suggest different approaches have advantages and disadvantages that are specific to the application domain. Where possible, they can be combined and provide better solutions than used as stand-alone tools. *Silicomp Research Institute* implemented the Script MIB on a small networked device [29]. They extended the implementation slightly to accommodate some needs they wanted related with time control, inter-script dependencies and script version information. On the other hand, the *Ja*va *S*cript *M*IB *I*mplementatio*n* (*JASMIN*) project [19], developed by the *Technical University of Braunschweig* and the *NEC C&C Research Laboratories Berlin*, implements the IETF Script MIB. Again, two extensible SNMP agents provide the support for the Script MIB: the commercial *SNMP Research Inc*. 's Enhanced MANagement Agent Through Extensions (*EMANATE*) SNMP agent toolkit [48] and the free-available *UCD-SNMP* agent [54] from *University of California at Davis*. They have analyzed [51] the time and memory consumption on the most frequently used operations, e.g. installing, starting, suspending, resuming and deleting a script, etc.; they have also estimated the time spent by the Java runtime engine's operations measuring the response time of the SMX commands.

2.3 Extensible Agents

Dynamic extensibility has been used to support the MbD model in network management environments as an alternative to overcome the limitations derived from centralized paradigms. Extensible agents are MbD agents that, upon requests by other management entities, they incorporate or delete managed objects and management services at run-time.

One approach based on the extensibility of management entities is the work being done by *Sun Microsystem Inc*. and other leading companies in the development of a set of standard specifications to equip the Java technology with a universal *extension for distributed management*, the Java Management eXtension (*JMX*) [52].

The JMX specifications propose a distributed management architecture based on manageable resources, a dynamically extensible agent and distributed management applications as shown in Figure 2.3. The JMX architecture is divided into three levels:

- the instrumentation level,
- the agent level and
- the manager level.

The instrumentation level specification provides the means to implement *JMX manageable resources* (managed resources), through a type of Java object that implements specific management interfaces known as Managed Bean (MBean for short). MBeans can be standard or *dynamic*, meaning that they expose the management interface at run-time. Besides, JMX proposes a notification mechanism for MBean instances to send notifications to components of the other levels.

The agent level provides the JMX agent specifications, which control the resources and make them available to the management applications. The JMX agent is a management entity composed of a MBean server, a set of agent services and at least one protocol adaptor or connector, which provides the communication services. The MBean server is the central registry for MBeans in the agent: all manageable resources wanting to be managed must be registered as a MBean in the server and all management operation requests for MBeans will go through this server. Agent services are also MBeans that perform management operations on other MBeans, for instance, location services, dynamic class loading, monitoring certain threshold values and notifying the change to others, a scheduling mechanism.

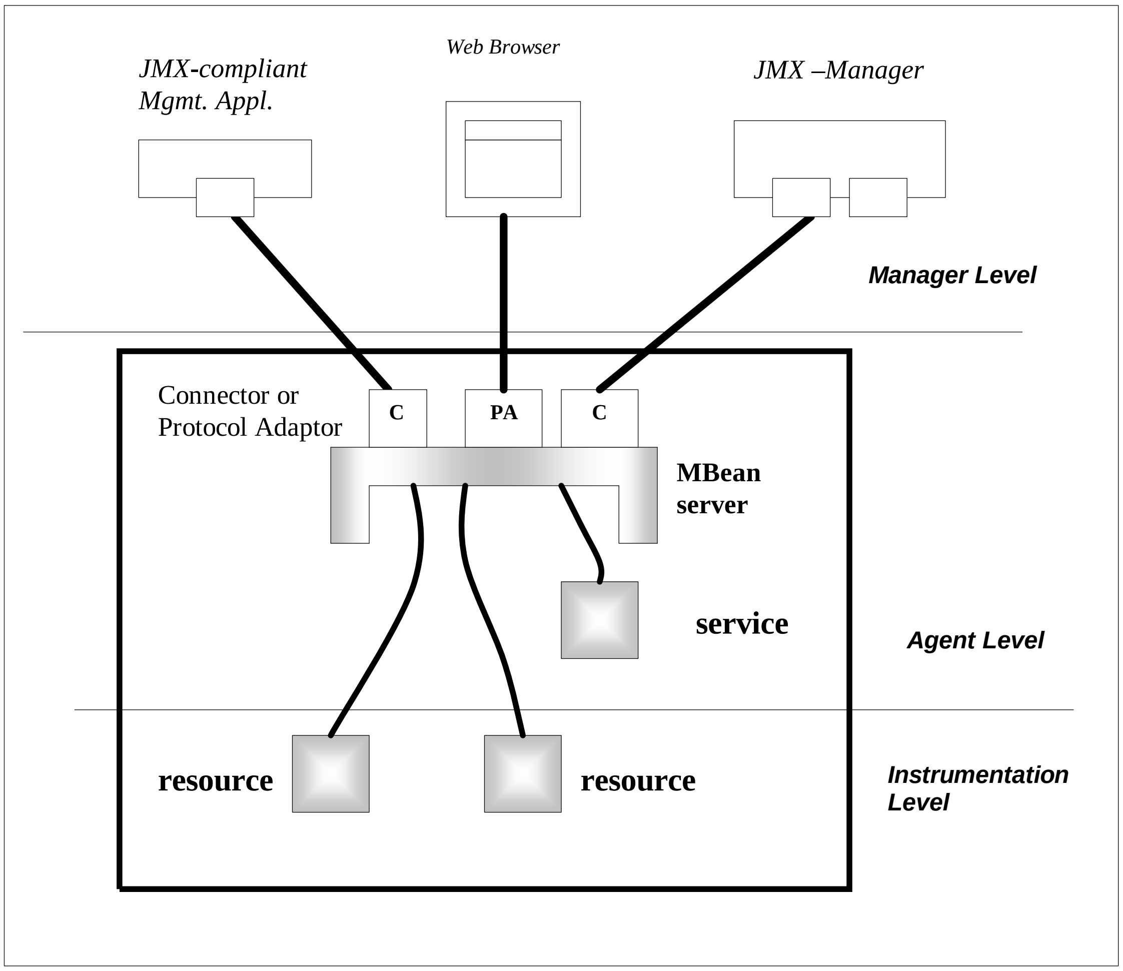


Figure 2.3 JMX Architecture

Although the manager level is not included in the present phase of specifications, the idea is that management applications can request management operations to a JMX agent for getting and setting MBean attribute values and performing operations on MBeans. Management applications can also get notifications from MBeans, instantiate, and dynamically register new MBeans with the JMX agent

In addition, it provides two management protocol APIs to integrate JMX compliant management applications with management systems: the SNMP manager API and the CIM/WBEM manager and provider API. They are sets of Java classes, which allow JMX-enabled management applications to access SNMP agents and CIM/WBEM Object managers respectively.

The second approach is based on e*xtensible agents* as MbD agents that model a distributed MIB in a master/subagents architecture where the MIB consists of a *static* MIB residing in the master agent and several distributed subagents that *dynamically* register portions of the MIB and interface the real managed objects. Extensible master agents require a kind of protocol interface to extend their capabilities to receive requests from the subagents to dynamically *regiter*/*unregister* MIB variables and, of course, to delegate to the subagents the appropriate requests for the MIB variables from the network manager side.

2.3.1 Extensibility implementations

The *EMANATE* architecture consists of an extensible Master Agent, a two part Application Programming Interface (API) and several subagents that can be developed and dynamically connected to the Master Agent at run-time. The SNMP Master Agent is the centerpiece of the *EMANATE* architecture that contains the SNMP engine supporting the three SNMPv1, SNMPv2c, and SNMPv3 versions of the management protocol; it manages the SNMP agent/subagent communication for registration, deregistration and lookup services; Master Agent provides multi-threading support for handling concurrent incoming messages asynchronously. *EMANATE* allows third parties to develop MIB-extending subagents that can be started and stopped independently of the Master Agent or other subagents. Master Agent allows different subagents to register the same MIB objects with the result that different rows of the same tabular object can reside in different subagents and the master deals with them in a priority based mechanism. *EMANATE* portability and efficiency has been based on the two part API: the system-independent API provides application developers with an easy way to port applications over different platforms and the system-dependent API optimizes the use of the resources in the underlying operational environment.

The public *UCD-SNMP agent* from University of California at Davis is another extensible SNMP agent, originally based on the Carnegie Mellon University SNMP (*CMU-SNMP*) implementation. An analysis of the two *CMU-SNMP* and *UCD-SNMP agent* data models can be found in [54] by the virtue of being the basis of the **Perpetuum Mobile Procura** project's *XMS-SNMP agent* design. The *UCD-SNMP agent* v 4.1.1 has been released recently.

2.3.2 The XMS-SNMP agent

The eXtended Mobility Supported SNMP agent (*XMS-SNMP agent*) [57, 37] of the **Perpetuum Mobile Procura** project is an extensible SNMP agent supporting an advanced network management model using mobile agents. The *XMS-SNMP agent* provides support for:

1. mobile agents arriving at the same network element (NE) of the agent and acting in the role of subagents (mobile subagents) extend the existing SNMP MIB tree, and

2. mobile agents arriving at the same network element (NE) of the agent and having management capabilities request data from the existing SNMP MIB tree.

2.3.2.1 The XMS-SNMP Agent Functional Architecture

Figure 2.4 shows the functional architecture of the *XMS-SNMP agent*.

The *XMS-SNMP master agent* uses the lightweight SNMP Distributed Protocol Interface (DPI) as the protocol supporting communication between the master agent and subagents that, taking advantage of the master agent and subagents locality, do not deal with ASN.1 details and BER encoding rules as the SNMP protocol does.

The DPI interface allows resident mobile agents to dynamically *register*/*untegister* MIB variables at run-time requiring neither re-compilation nor re-starting the agent nor forcing the agent to re-read a configuration file.

In a typical scenario, an arriving mobile agent subagent extending the MIB sends a DPI *open* request to establish a communication link with the *XMS-SNMP agent*. Then, the subagent requests the registration of the MIB groups it supports. Once the registration process is complete, the subagent waits for requests from the *XMS-SNMP agent* for accessing the extended MIB objects. Occasionally, the subagent sends trap the agent receives and forwards them to an SNMP manager. Either the subagent or the *XMS-SNMP agent* can *unregister* the extended MIBs and can *close* the connection at any time.

The *XMS-SNMP agent* allows multiple subagents to register MIB sub-trees simultaneously; a subagent can register multiple MIB sub-trees and, a MIB sub-tree can be registered by multiple subagents.

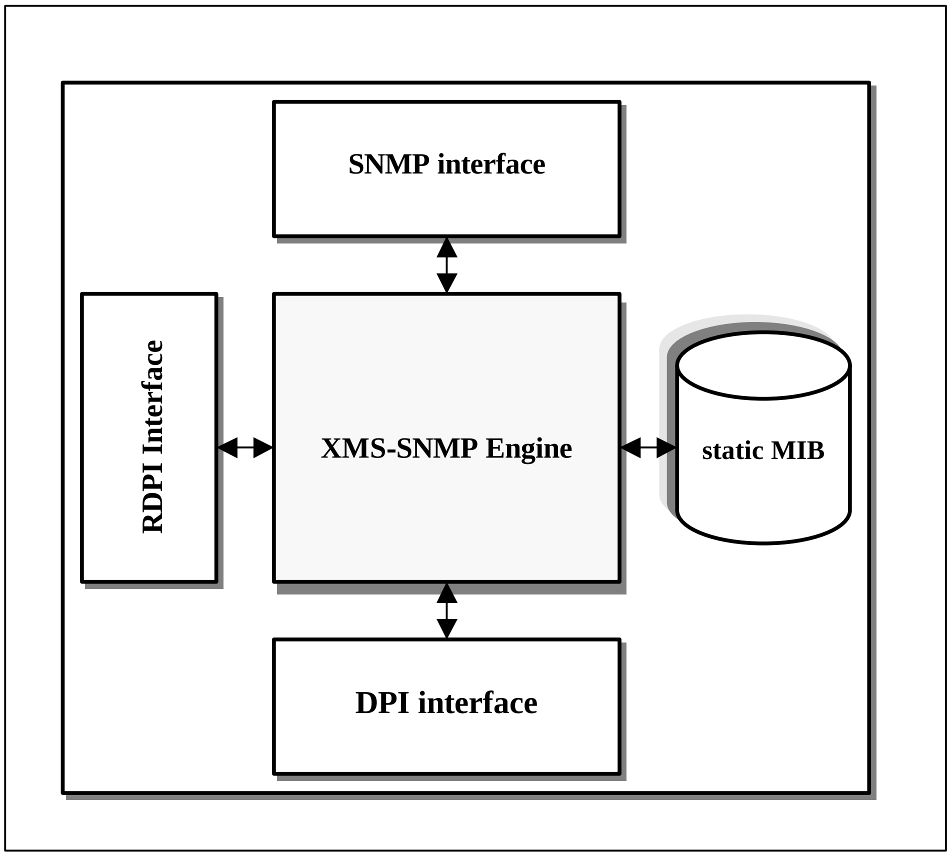


Figure 2.4 The XMS-SNMP Agent Functional Architecture

The lightweight SNMP Reversed DPI (RDPI) interface is used by the *XMS-SNMP agent* to accept management requests from arriving mobile agents acting in the role of a manager and wishing to access MIB variables, and the SNMP interface to communicate with remote manager applications via the SNMP protocol. Taking advantages of the local communication, the RDPI protocol processes incoming requests and creates responses more efficiently. The RDPI protocol is fully described and a comparative analysis of RDPI and SNMP can be found in [57] and [37] . In section 6.2 of this thesis the average response time of management operations is measured and shows how much faster is RDPI in decoding packets than SNMP.

The XMS-SNMP Engine dispatches management requests from these interfaces, providing access to the MIB variables including the extensions currently provided by subagents via DPI.

2.3.2.2 The XMS-SNMP Agent Data Model

The basis of those multi-registration capabilities resides in a dynamic three-dimensional data model and how the *XMS-SNMP agent* schedules the processing of messages from different interfaces.

Figure 2.5 shows the *XMS-SNMP agent* data model used to organize the MIB-sub-trees registered with the agent, the list of subagents registering each MIB sub-tree and, the list of MIB sub-trees registered by each subagent.

1. A double linked header, *subtreeList*, points to a double linked list of *subtree* structures. Each structure represents a static (built-in) or dynamic MIB sub-tree.

2. If dynamically registered, the MIB sub-tree structure has a pointer to a double linked header, *regTreeList***,** pointing to a double linked list of *registerTree*

structures. There is one *registerTree* in the list for each subagent registering this MIB sub-tree. Each *registerTree* structure contains the parameters with which the subagent has registered this MIB sub-tree and a pointer to the *subagent* itself.

3. The *subagent* structure has a pointer to *regTreeSet*, an array of pointers to all *registerTrees* registered by the subagent with the *XMS-SNMP agent*.

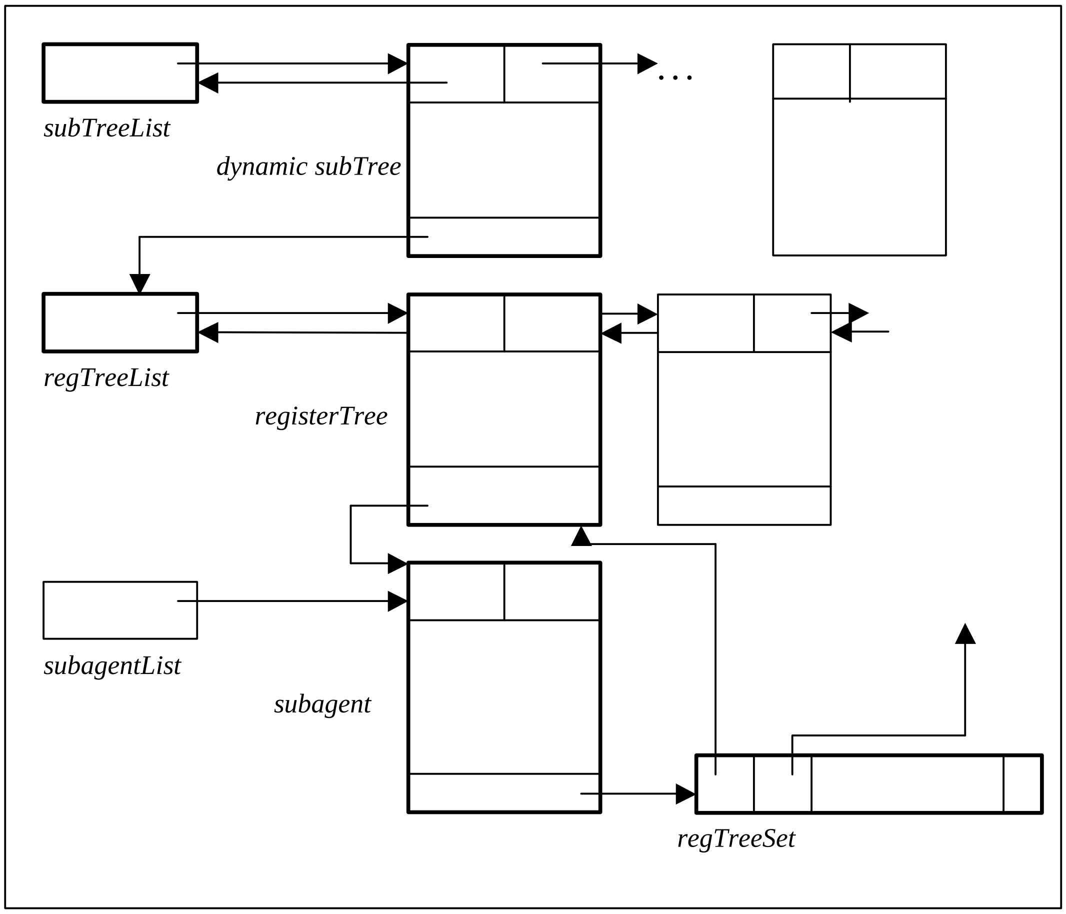


Figure 2.5 The XMS-SNMP Data Model

2.3.2.3 The XMS-SNMP Agent Message Scheduling

The *XMS-SNMP agent* receives and schedules SNMP messages as well as DPI and RDPI messages.

Extending the XMS-SNMP agent's MIB

A subagent happens to send a DPI message to *open* or *close* a connection, and to *register* or *unregister* MIB sub-trees with the *XMS-SNMP agent*.

If a DPI *open* message is received, the *XMS-SNMP agent* simply creates a *subagent* structure and concatenates it in the *XMS-SNMP agent*'s *subagentList*.

If a DPI *register*-a-MIB-sub-tree message is received, the *XMS-SNMP agent*

- searches in the *subagentList*;
- creates a *subTree* structure if needed, and inserts it in the *subTreeList*,
- creates the *registerTree* structure for the MIB sub-tree being registered by this subagent, adds it to the subagent's *regTreeSet* and,
- inserts the *registerTree* into the MIB sub-tree's *regTreeList*.

The *registerTree* structure is built with a priority value argument and the *regTreeList* is maintained in a priority order ranging from 1 to 128; the highest priority the lowest value is first. The priority order is determined as follows:

a) If the *registerTree* to be inserted comes with priority value equal to –1, it will try to register the MIB sub-tree with the highest priority, if available. If succeeds, the new subagent registering the MIB sub-tree becomes its *authoritative* subagent.

b) If the *registerTree's* priority value is equal to zero, it will try to register the MIB sub-tree with the next to its current highest priority. If succeeded, the new subagent registering the MIB sub-tree becomes its *authoritative* subagent.

c) In other cases, it will try to register the MIB sub-tree with the specified or next higher priority.

Accessing the MIB

The *XMS-SNMP agent* receives and schedules SNMP and RDPI requests for management applications.

Either an SNMP or an RDPI request, for each MIB variable in the variable-binding list coming in the received message, the variable sub-tree ID[3] is extracted from its object identifier (OID) and it is searched through the *subTreeList*. If it succeeds and the sub-tree is dynamically registered, the agent creates a DPI

[3] An object identifier (OID) usually comprises a sub-tree ID and an instance ID.

message, sends it to the *authoritative* subagent and waits for the response to forward it to the requesting application.

2.3.2.4 The XMS-SNMP Management MIB

The *XMS-SNMP agent* also allows SNMP-based management of the extended platform itself. Management information is available and organized into the hard-wired *xmsAgentMIB* MIB group that provides SNMP-based monitoring and control capabilities of mobile agent subagents and their dynamically registered MIB sub-trees.

Figure 2.6 displays the position in the OSI registration tree and the organization of the *xmsAgentMIB* MIB group and the *subagentMIB* and the *registerTreeMIB* subgroups as defined in [57].

The *subagentMIB* subgroup comprises the *subagentNumber* variable and the *subagentTable* tabular object. The *subagentNumber* represents the current number of subagent entries in the table and, each subagent entry in the *subagentTable* has a number of variables describing the attributes of a mobile subagent. Each subagent entry has the current number of sub-trees that the subagent has registered. It also has a variable called *subagentAdminStatus*, for allowing remote network manager to *close* the agent-subagent connection.

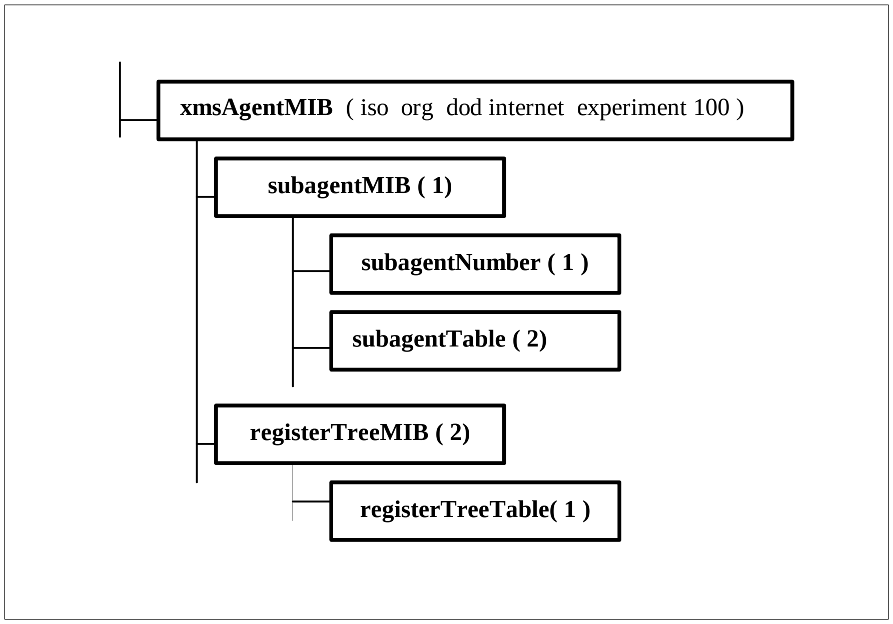


Figure 2.6 The XMS-SNMP Agent Management MIB

If an SNMP or RDPI *set* request is received to change the *subagentAdminStatus* variable to 1, the agent will:

- create and send a DPI *close* message to the subagent,
- remove the subagent from the *subagentList*,
- remove every *registerTree* in the subagent's *regTreeSet* from the corresponding *regTreeList*,
- if any of the *regTreeList* is empty, remove the corresponding *subtree* from the *subtreeList*

The *registerTreeMIB* subgroup comprises only a tabular object called the *registerTreeTable*. Each entry in the table describes the attributes of a *registerTree* (see subsection 2.3.2.1 above) registered by a subagent. The *registerTreeAdminStatus* is the variable for allowing network manager to *unregister* a *registerTree*.

If an SNMP or RDPI *set* request is received to change the value of the *registerTreeadminStatus* of a *registerTree*, it will:

- create and send a DPI *unregister* message to the subagent,
- remove the *registerTree* from the *registerTreeList*,
- if the *registerTree* is empty, remove the *subtree* as well, and
- nullify the pointer in the subagent *regTreeSet*.

2.4 Summary

Delegation techniques have been proposed to address the problems that arise in network management systems because of the rapid growth, heterogeneous environment, and geographical and administrative distribution facing today's telecommunication networks. This is the case with the *XMS-SNMP agent* of the **Perpetuum Mobile Procura** project, which implements an extensible SNMP agent with the purpose of integrating a mobile agent environment into an SNMP-based network management system.

Mobile agent technology brings new opportunities for decentralization in network management systems that the delegation of management scripts model does not. In the DISMAN Script MIB model the network manager always triggers the code migration, and there is no mechanism for autonomous mobility for these delegated scripts. The DISMAN framework suggests the manager can delegate management functionality along a pre-established (static) hierarchy of SNMP entities comprising the overall management system. However, mobile agents can also follow a pre-established migration path or alternatively, they can implement the migration path of their own or ultimately can select its migration path heuristically, allowing data collection and processing closer to the originator; providing a better use of network bandwidth reducing management traffic; enhancing flexibility and adaptability when coping with unexpected (and unwanted) breakdowns and non-permanent links; etc. Mobile agent technology also proposes inter-agents and mobile agent-management systems communication mechanisms network protocol-independent but the communication capabilities in the script MIB model are based only on HTTP, FTP and SNMP.

Therefore, MbD can benefit from the advent and development of the mobile agent technology. The next chapter discuses the mobile agents capabilities for network management.

Chapter 3 Mobile Agent Environment for Network Management

3.1 Introduction

This chapter makes a comparative review of two standardization initiatives for agent systems *interoperability*, the Object Management Group (OMG) standard known as the Mobile Agent System Interoperability Facilities (*MASIF*) and the Foundation for Intelligent Physical Agents (FIPA) standards, *FIPA specifications*. The mobile agent framework discussed here is the *Mobile Code Toolkit* (MCT), the mobile agent infrastructure of the **Perpetuum Mobile Procura** (PMP) project. PMP is devoted to advanced network management using mobile agent technology.

The advent of new technologies like the OMG's Common Object Request Broker Architecture (CORBA) [34] and mobile agents have created new potential for the development of scalable, heterogeneous and distributed network management systems.

The distributed object architecture paradigm adopted by CORBA is one of the approaches to handle the complexity and heterogeneity of network management systems. One of the key advantages of CORBA is that it provides open interfaces for integration of existing management solutions, distributed management components communicate exchanging management information using these standardized interfaces over networks consisting of devices from diverse vendors.

Its distribution capabilities make CORBA-based solutions to manage large number of network devices in a scalable manner. Its object-oriented paradigm proposes a higher level approach simplifying the development of distributed management services. Location-transparency object manipulation, and language and operating system independence are also characteristics CORBA-based solutions can provide to network management systems.

Particularly for managing large scale, heterogeneous and dynamic telecommunication networks, the use of CORBA has being focused on interoperability with the OSI Systems Management (OSI-SM) framework [39]. As in SNMP, the architecture of the OSI-SM for the ITU-T Telecommunication Network Management (TMN) [39], is based on the manager-agent paradigm and the information being specified uses the Guidelines for the Definition of Managed Objects (GDMO) [39]. The object location and representation transparency, and the relatively ease to learn and use are CORBA properties that benefit distributed TMN system requirements, complemented by the OSI-SM as the technology for management information exchange. The OMG's Joint_Inter-Domain Management (JIDM) [20] group has been working on the mapping between GDMO/ASN.1 to CORBA IDL, and between SNMP SMI and CORBA, the specification of generic gateways between different management technologies and the JIDM agent as the architecture of a management agent (TNM agent) that defines objects as individual CORBA objects and where the relations between object instances are also modeled using CORBA references.

However network management systems are also interested in dynamic behavior, modifying dynamically the management capabilities of the network elements and in managing distributed resources that might request location information, instead of having predictable management functionality through static object platforms such as CORBA does. Then, the paradigm of moving management logic closer to the management data when required is considered.

Mobile agent is another essential technology for the development of distributed software applications because of the capabilities of mobile agents to move across distributed environments, integrate with local resources and other mobile agents, and communicate their results back to the user or authority on behalf they are acting. Particularly in the field of telecommunications, the decentralization of management functionality benefits from their integration with the mobile agent technology.

Their cooperative, autonomous and migratory nature make mobile agents better suited to overcome the limitations the SNMP framework imposes even when integrated with other distributed design technologies; for example, with the Script MIB model or CORBA. Mobile agents can be equipped with specialized management tasks and then deployed to the network elements. In performing their tasks, mobile agents migrate autonomously from one NE to another, as close to managed resources as possible, access the managed data; and operate independently, minimizing the manager interaction. Because of the ability of mobile agents to communicate with each other, they can cooperate with other exchanging data and logic, bringing management system capabilities to distribute management tasks in a composition of small, collaborative, and intelligent mobile agents. Management mobile agents can provide network manager applications with a suitable level of abstraction; the way the managed data are organized and the operations to access them are wired into the management mobile agents, frees the network manager to be aware of these low level operative tasks.

By virtue of the increasing availability of agent platforms, an important goal in agent technology is the interoperability between different agent systems, meaning the cooperation among agents from different agent platforms of different manufacturers. These two standardization bodies related to agent technology are addressing interoperability of agents from different manufacturers in order to fulfill the requirements of today's dynamic, heterogeneous and distributed service provisioning applications. Section 3.2 reviews the OMG's MASIF and the FIPA specifications. Section 3.3 presents the functional architecture of the PMP project's mobile agent platform, the *Mobile Code Toolkit* (MCT). It also presents the mobile agent management capabilities, the location services and migration history control in a MCT environment.

3.2 Initiatives for Agent System Interoperability

An agent platform is a software environment in which software agents run [14]. It provides support for software agents to execute, to manage their execution, to access system resources, and to guarantee integrity and protection of agents and the platform itself. Agent platforms also provide support for migration, naming, location and communication services.

Although developed for general or specific purposes, at the top of different host operating environments, using different software design technologies, the current variety of different agent platforms is exhibiting certain common trends. They provide Java-based agent environments and platform services based on middleware like CORBA IIOP and Java’s RMI where communication and migration capabilities are built at the top of them.

MASIF agent interoperability is based on agent platforms that providing the same programming language environments, the agent system type, and the serialization and authentication mechanisms, enable mobile agents to *migrate* from one to another. FIPA specifications agent interoperability is based on remote *communication* services.

3.2.1 MASIF and FIPA similarities

For both, the Object Management Group's Mobile Agent System Interoperability Facilities (MASIF) standard [36] and the Foundation for Intelligent Physical Agents (FIPA) specifications [9, 10], the main goal is to establish the common basis for heterogeneous agent platforms to enable interoperability amongst them.

Figure 3.1 and Figure 3.2 show the MASIF and FIPA agent reference models respectively. Some structural and functional similarities have been identified in both technologies.

1. MASIF *region* and FIPA *domain* represent a set of distributed and co-operating agent platforms that belong to the same authority. In these contexts, they are regarded as security domains.

2. MASIF agent system (*agency*) and FIPA *Agent Platform* (AP) are the software systems where agents reside and execute.

3. MASIF *place* and FIPA *domain* can be compared in the sense that they group agents in a logical execution environment. An agency supports multiple places and a FIPA AP supports multiple domains (a domain can also comprise multiple APs).

4. The MASIF *MAFAgentSystem* interface and the FIPA Agent Management System (*AMS*) component provide the mechanisms for managing the life cycle of agents executing in the platform. They define the management operations to create, suspend, resume, terminate, and migrate agents.

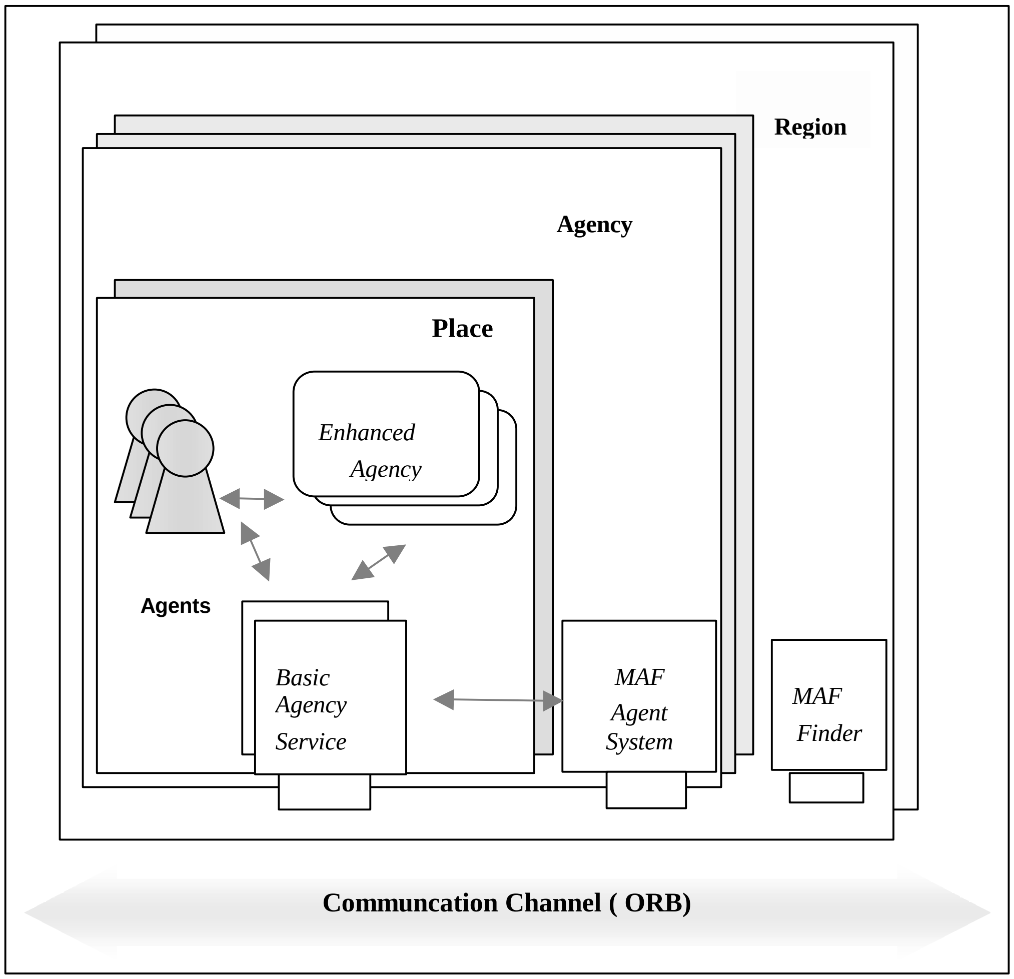


Figure 3.1 The MASIF Architecture

5. The MASIF *MAFFinder* interface and the FIPA Directory Facilitator (*DF*) component provide the methods for maintaining dynamic registration services.

6. The MASIF *MAFFinder* interface and the FIPA *AMS* define the naming and location directory.

7. The MASIF Communication Channel and the FIPA Agent Communication Channel (*ACC*) address the communication facilities.

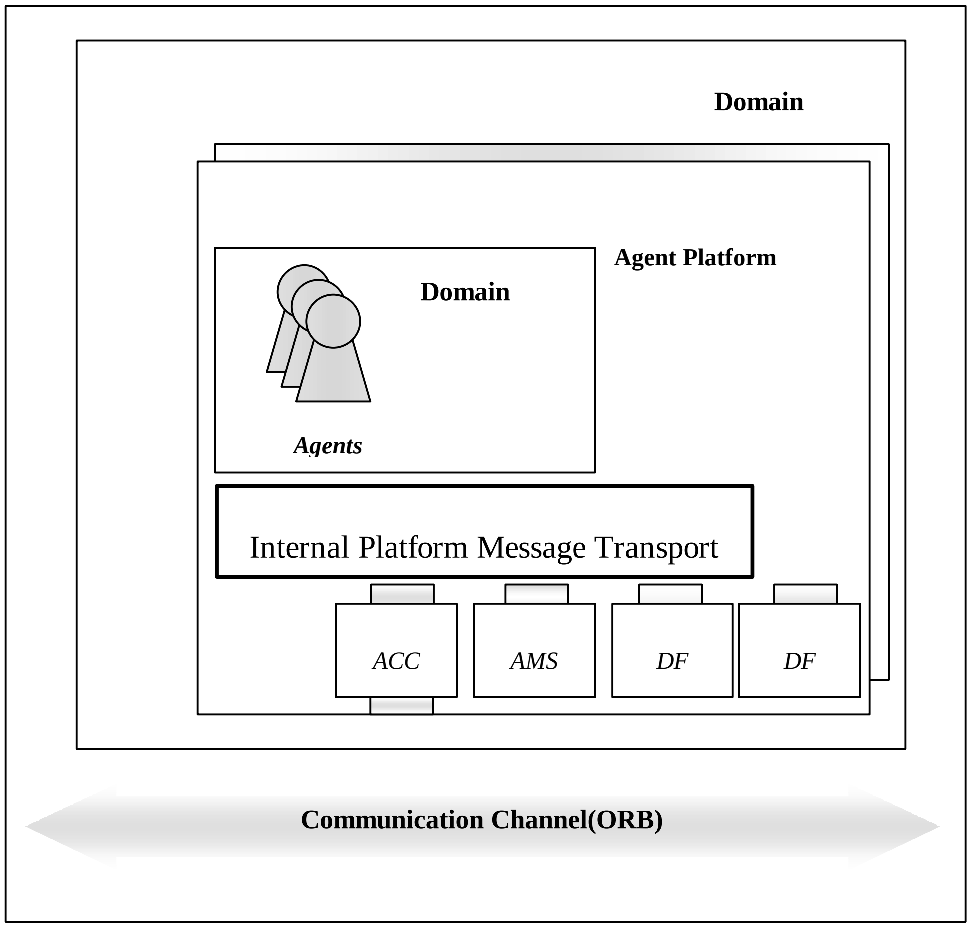

Domain
Agent Platform
Domain
Agents
Internal Platform Message Transport
ACC
AMS
DF
DF
Communication Channel(ORB)

Figure 3.2 The FIPA Reference Model

3.2.2 MASIF and FIPA differences

Nevertheless, OMG MASIF work is primarily based on *mobile agents* (mobile agent) traveling amongst agent systems of the same profile[4] via the CORBA IDL interfaces and does not address the inter-agent communication at all. On the other side, FIPA specifications focus on *intelligent agent* (IA) communications via content languages and do not say much about mobility.

Within a mobile agent paradigm, the co-operation in distributed dynamic environments is realized through the encapsulation of the delegated functionality into the mobile agents and deploying them to the network where they can best perform their tasks, thus taking advantage of the local communication. MASIF-compliant platform functionality is accessible via two CORBA IDL interfaces, the *MAFAgentSystem* and the *MAFFinder*. They are defined at the agent system level rather than at the agent level to address the interoperability concerns. Figure 3.3 shows a typical sequence of an agent looking for the destination agent with which it wants to communicate.

On the other side, within an agent communication paradigm, IAs (mostly static) co-operate via the Agent Communication Language (*ACL*), the *content language* based on predicate logic and the *ontology* which identifies the set of basic concepts used in the message content for co-operation. In FIPA specifications, all agents have access to at least one Agent Communication Channel (*ACC*) that support the path for basic contact and interchange among agents, including the DF and AMS. Inter-platform communication takes place via an ACC, being CORBA IIOP the recommended default communication protocol.

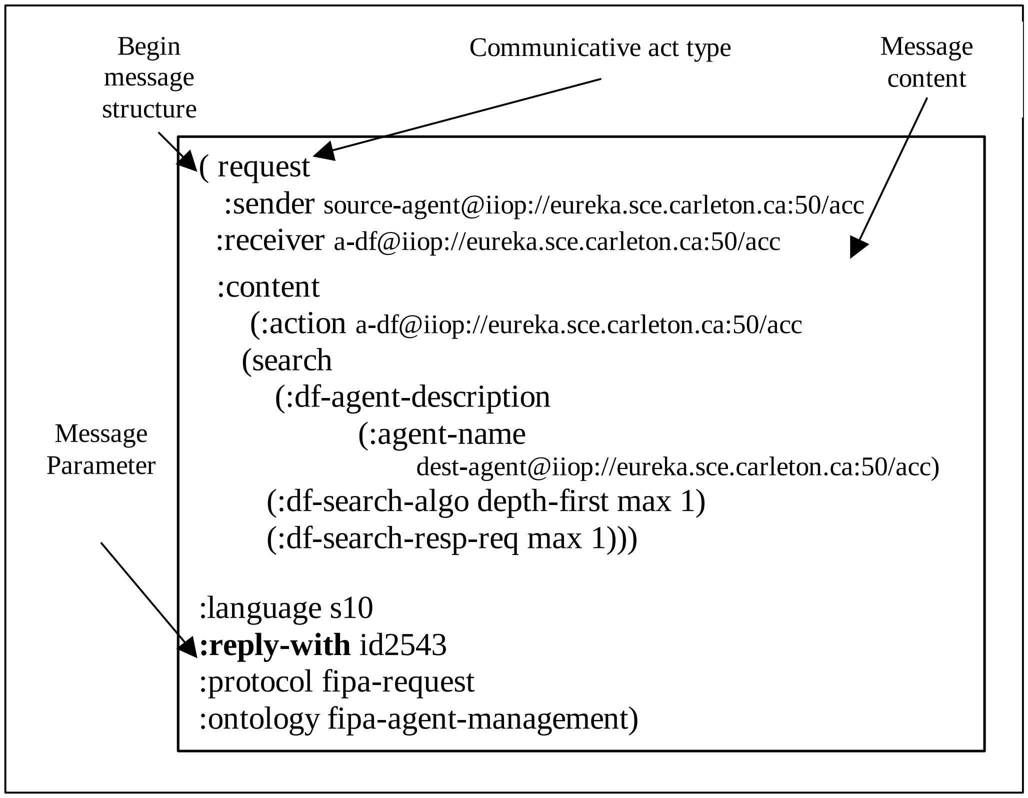
```
// . . .
mafFinder mFinder;
try {
 // get the mobile agentFFinger reference
mFinder = agentSystem.get_mobile agentFFinder();
} catch (FinderNotFound fe) {
  System.out.println("mobile agentFFinder not available");
 exit();
}
Location destLocation;
Name destAgentName = new Name("public", "destAgentID", 0);
try {
destLocation = mFinder.lookupAgent(Name, null);
} catch (EntryNotFound e) {
  System.out.println( destAgentName+" not found in region");
exit();
}
// destLocation specifies the agent system where the agent
// resides currently. It must be converted to the object reference of the
// agent system.
```

Figure 3.3 Finding a Destination Agent

The FIPA *ACL* [9] provides an interface for exchanging messages among agents. It derives from the Knowledge Query and Manipulation Language (KQML) [55]. It is based on speech *act theory*: messages are actions, or *communicative acts*, as they are intended to perform some actions by virtue of being sent. Figure 3.4 is an example of an FIPA ACL message.

[4] An agent profile refers to the agent system type, programming language, authentication and serialization methods.[34]

Ontology is an explicit specification of some topic. It includes a vocabulary of terms in the subject area, the integrity constraints on these terms, and the logical statements describing their meanings and how they are related to each others. The FIPA Ontology Service specification [10] deals with technologies for definition and management of ontology.



```
( request
   :sender source-agent@iiop://eureka.sce.carleton.ca:50/acc
  :receiver a-df@iiop://eureka.sce.carleton.ca:50/acc
  :content
      (:action a-df@iiop://eureka.sce.carleton.ca:50/acc
     (search
         (:df-agent-description
                 (:agent-name
                       dest-agent@iiop://eureka.sce.carleton.ca:50/acc)
        (:df-search-algo depth-first max 1)
        (:df-search-resp-req max 1)))

:language s10
:reply-with id2543
:protocol fipa-request
:ontology fipa-agent-management)
```

Figure 3.4 FIPA ACL Message

The FIPA Agent Management Support for Mobility specification [10] proposes the minimal set of technologies required for supporting agent mobility using the FIPA agent platform. The reference model recognizes there are different forms to express mobility, such as code mobility, agent migration, and agent cloning. FIPA provides a set of primitive actions to support extensible forms of mobility protocols, including *move*, *transfer*, *execute* and *terminate*.

The agent communication paradigm adopted by FIPA supports a higher level of interoperability between heterogeneous systems given that MASIF interoperability is based on agents systems written in the same programming language.

The agent communication paradigm can better express the nature of co-operation and is more suitable for integration with other artificial intelligent technologies; therefore, it will be useful in a wider range and more complex applications.

The mobile agent paradigm can be more appropriate in situations where dynamic and autonomous swapping, replacement, modification, and updating of application components are required.

Both the mobile agents and the agent communication paradigms have advantages and disadvantages that are application-domain specific, such as the applicability of MASIF-compliant or FIPA-compliant agent systems. Therefore, some important issues arise regarding these two agent-based distributed software design technologies in the context of telecommunication applications. For instance, can the prevalence of either MASIF or FIPA or both technologies be envisaged? Would it be desirable to combine the two technologies? How can MASIF and

FIPA technologies be combined into a unified mobile agent framework? How can MASIF and FIPA inter-operate?

Annex A of the FIPA 98 Part 11 specification [10] makes a proposal with four variants in order to realize a MASIF- and FIPA-compliant agent platform. The first two variants are based on modifications to the existing *MAFAgentSystem* and *MAFFinder,* the MASIF standard interfaces. The third variant proposes the extension of the MASIF standard with new interfaces to achieve the FIPA specifications in the unified agent platform. The fourth one argues for the provision of low-level methods to the FIPA specifications as defined in MASIF standard.

Annex C of the ACTS[5] baseline document [28] analyses the possibilities of MASIF/FIPA integration. The *IAs with intelligent mobility* approach extends the concept of "message agent", considering mobile agents as a special kind of agent communication message content, meaning ACL containers will identify the type of content as mobile agent with a particular agent profile. The *MASIF/FIPA interoperability via a IDL/ACL Gateway on mobile agent Agent System* uses an IDL/ACL gateway installed in a dedicated MASIF agent system to enable the co-operation between non-ACL mobile agents and remote IAs. The *mobile agent with Intelligence* approach adds another facility to the MASIF environment: a

[5] ACTS is an European research collaborative program for Advanced Communications, Technologies and Services [ACTS99].

FIPA ACC for ACL communication, suggesting the implementation of ACL speech acts in IDL to maintain the lightweight feature of the mobile agents.

3.2.3 MASIF and FIPA implementations

Concerning agent platform development, the IKV++ ’s *Grasshopper* [18] is a mobile agent and runtime platform developed in Java, built upon a distributed object-oriented middleware and compliant with the OMG MASIF standard. Its Distributed Agent Environment (DAE) is composed of regions, agencies, places and agents.

From a MASIF perspective, in a *Grasshopper* DAE, the region provides management capabilities, location services and facilitates inter-agency communication for agents. All agencies and their respective places associate with a region as early as they are created by registering within the *region registry* and remain associated with this single region for all their lifetime. Agents are also registered with the registry and if mobile, the registry is updated each time mobile agents migrate from one agency in the region to another. A *Grasshopper* region implements the MASIF *MAFFinder* interface to provide other entities with the abilities for locating agents, places and agencies residing in the region.

Agencies consist of a *core agency* and various places. The core agency implements the MASIF *MAFAgentSystem* standard interface and provides management, registration, communication, persistence, communication, security, and transport services to the agent execution environment. Agents can be of two types, mobile or stationary.

Besides the CORBA-based MASIF standard interfaces, Grasshopper core agency's Communication Service (CS) provides agencies and agents with location-transparent remote communication using CORBA IIOP, Java's RMI, and plain socket connections and the last two can optionally be protected with the Secure Socket Layer (SSL). As shown in Figure 3.5, from the agent's point of view, there is no difference between remote and local method invocations by the virtue to the use of *proxy agents*. In addition, in a Grasshopper DAE, the inter-agent communication protocol is determined dynamically and since they are realized via plug-in interfaces, it can easily be expanded with other communication protocols.

Grasshopper is also a Java-based agent development platform. An agent is composed of one or more Java classes, the one of them that builds the actual core of the agent is specified as the *agent class*. They provide proprietary and MASIF complaint interfaces allowing agents to access to the Grasshopper DAE functionality, the local and remote agencies, and region registry.

Its management functionality is divided in three categories, including the agency, place and agent management that are available through a graphical interface known as the *Agency Console* and through its Application Programming Interface (*Grasshopper* API) as well. Management capabilities are summarized in Table 3.1

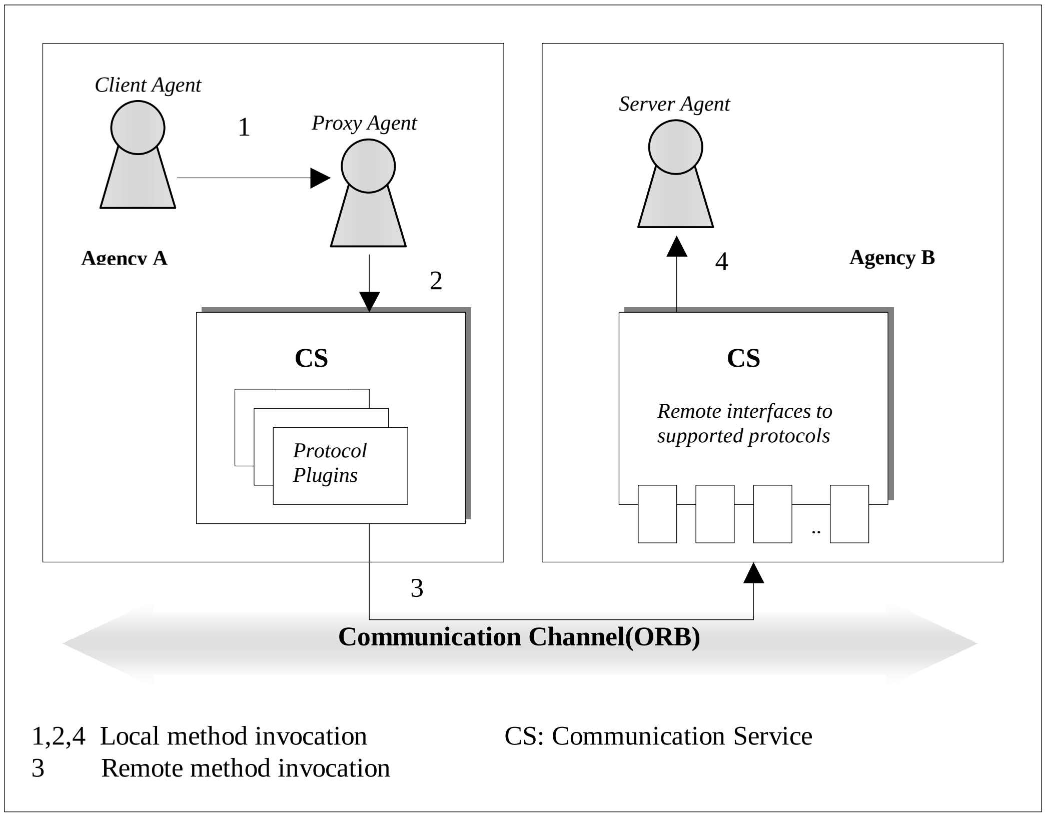

Client Agent
1
Proxy Agent
Agency A
2
CS
Protocol Plugins
Server Agent
4
Agency B
CS
Remote interfaces to supported protocols
..
3
Communication Channel(ORB)
1,2,4 Local method invocation
3 Remote method invocation
CS: Communication Service

Figure 3.5 Location Transparency in a Grasshopper DAE

IKV++ has announced the next generation of the mobile agent platform, *Grasshopper-2*, which integrates both MASIF and FIPA agent standards but there is not information available saying how far the environment is FIPA compliant.

	Categories		
	Agency	***Place***	***Agent***
	Agent Catalog	Creation	Creation
	Configuration of Preferences	Removal	Removal
	Monitoring events	Suspension	Suspension
Events	Thread Monitoring	Resumption	Resumption
	Agency termination		Cloning
			Copying
			Migration
			Saving
			Action invocation

Table 3.1 Grasshopper Platform Management

Another implementation comes from IBM. IBM's *Aglets* [16] are Java objects that can execute on a host, halt its execution, move to another host in the network and resume its execution there. The Aglet architecture consists of two layers: the *Aglet*s Runtime Layer and the Communication Layer.

The Communication API conforms to the MASIF *MAFAgentSystem* interface and it is possible to integrate not only with CORBA IIOP but also with Java RMI and the proprietary Aglet Transform Protocol (ATP). Actually, the current implementation supports ATP and RMI; CORBA transport layer is not supplied yet. The *MAFAgentSystem* interface is an abstract class that is implemented in two classes: one provides the agent system facility and the other is the protocol-dependent stub object. In this case, a client *aglet*, as shown in Figure 3.6, explicitly choose the corresponding stub object for a particular protocol to communicate with its destination.

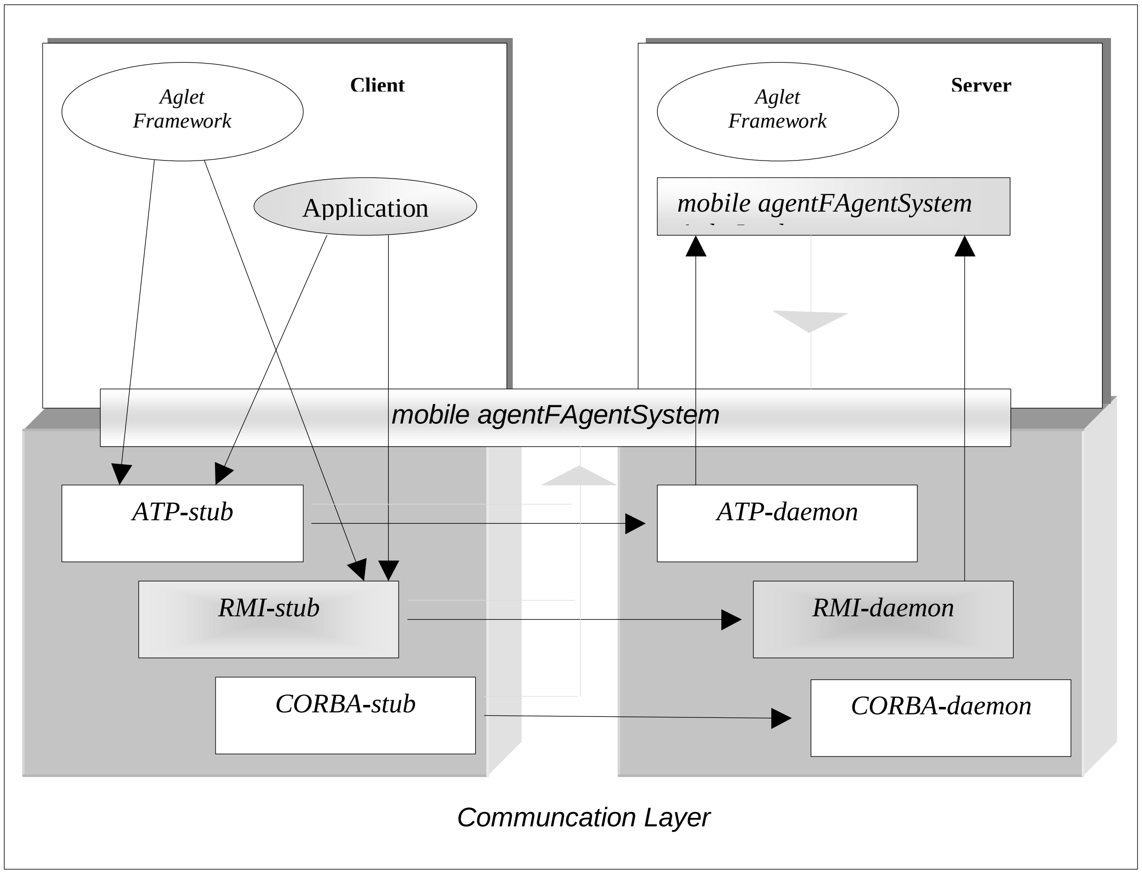


Figure 3.6 Aglet Communication Layer Architecture

The third example is the *FIPA-OS* from Nortel Networks [32]. The *FIPA-OS* is a publicly available FIPA compliant agent platform. FIPA-OS is an agent runtime and development platform using Java 1.2.2 JDK and has been tested on Windows 95, Windows NT, Linux and Solaris operating environments. In addition to the mandatory components of the FIPA Reference Model (see Figure 3.2), the FIPA-OS distribution provides a class hierarchy, the *Agent Shell*, for the development of FIPA-OS agents. As shown in Figure 3.7, the base abstract class *Agent*, provides the basic requirement of a FIPA-OS agent, the direct subclass *AgentWorldAgent* adds support for inter-agent and agent-platform communication and registration with local and remote platforms facilities are provided. The platform agents *AgentCommunicationChannel*, AgentManagementSystem, and the *DirectoryFacilitator* are implementation of the FIPA-OS agent shell.

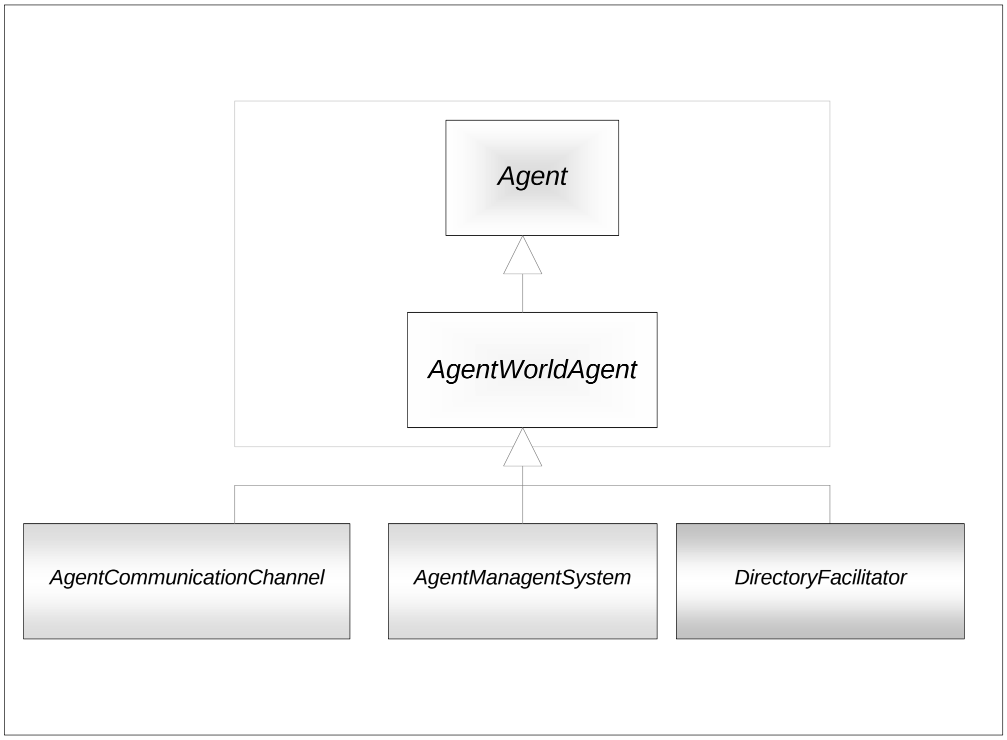


Figure 3.7 The FIPA-OS Agent Shell

The FIPA-OS's transport layer supports additional object request broker implementations, *Voyager ORB* from PTS [42] or from ObjectSpace [33] that simultaneously support CORBA and RMI. The content language and the agent profile support the Resource Description Framework (RDF) encoding[6] and therefore FIPA-OS has included the *SiRPAC* RDF parser from W3C [60] and the *Xercer* XML parser from Apache [2].

3.3 The Perpetuum Mobile Procura Project's Mobile Code Toolkit

The *Mobile Code Toolkit* (MCT) [40] of the PMP project is an infrastructure for the use of mobile agents in telecommunication network management.

The MCT is a lightweight and freely available [6] Java-based mobile agent environment. The current implementation provides mechanisms for communication and migration capabilities built on top of Java's RMI and client applications can benefit from the integration of the mobile agent paradigm with the RMI technology.

Figure 3.8 shows the MCT environment infrastructure. In a MCT environment (MCE), a Mobile Code Daemon (MCD) runs within a separated Java Virtual Machine (JVM) and, at the top of this daemon, there are a number of other components providing a number of services that provide mobile agents with an execution environment and facilitate the performance of their management tasks. The Mobile Code Manager (MCM) is the part in the MCD that implements the

[6] RDF is a W3C recommendation for describing and interchange metadata on the Web using the eXtensible Markup Language(XML) as the syntax descriptor [57,58].

mobile agent’s life cycle model. It maintains control of all mobile agents instantiated within the MCE and can create, start, suspend, resume, stop, and destroy mobile agents. The Communication Facilitator (CF) is the component that allows a mobile agent to collaborate with other mobile agents and provides the support for local and regional location services. In an MCE, the Migration Facilitator (MF) is the part of the MCD that provides the infrastructure with code migration capabilities.

In addition, MCT provides a standard interface, the Virtual Managed Component (VMC), through which mobile agents can gain controlled access to local resources. A VMC is a stationary agent on its Network Element (NE), which encapsulates the underlying characteristics and provides standard access and allocation services of local resources to mobile agents. First, their stationary nature provides the MCM with a controlled way to manage how local resources have being used. Secondly, preventing mobile agents to gain direct access to resources provides security to the MCE and allows lightweight mobile agents as well. mobile agents wanting to access local resources can obtain handles to their respective VMCs from the MCM.

3.3.1 Mobile Agent Management

The MCM is the responsible to maintain the mobile agents visiting the MCE.

Upon arrival and after performing the authentication checking the MCD allows a mobile agent to run on the MCE. The MCM then instantiates and registers the mobile agent.

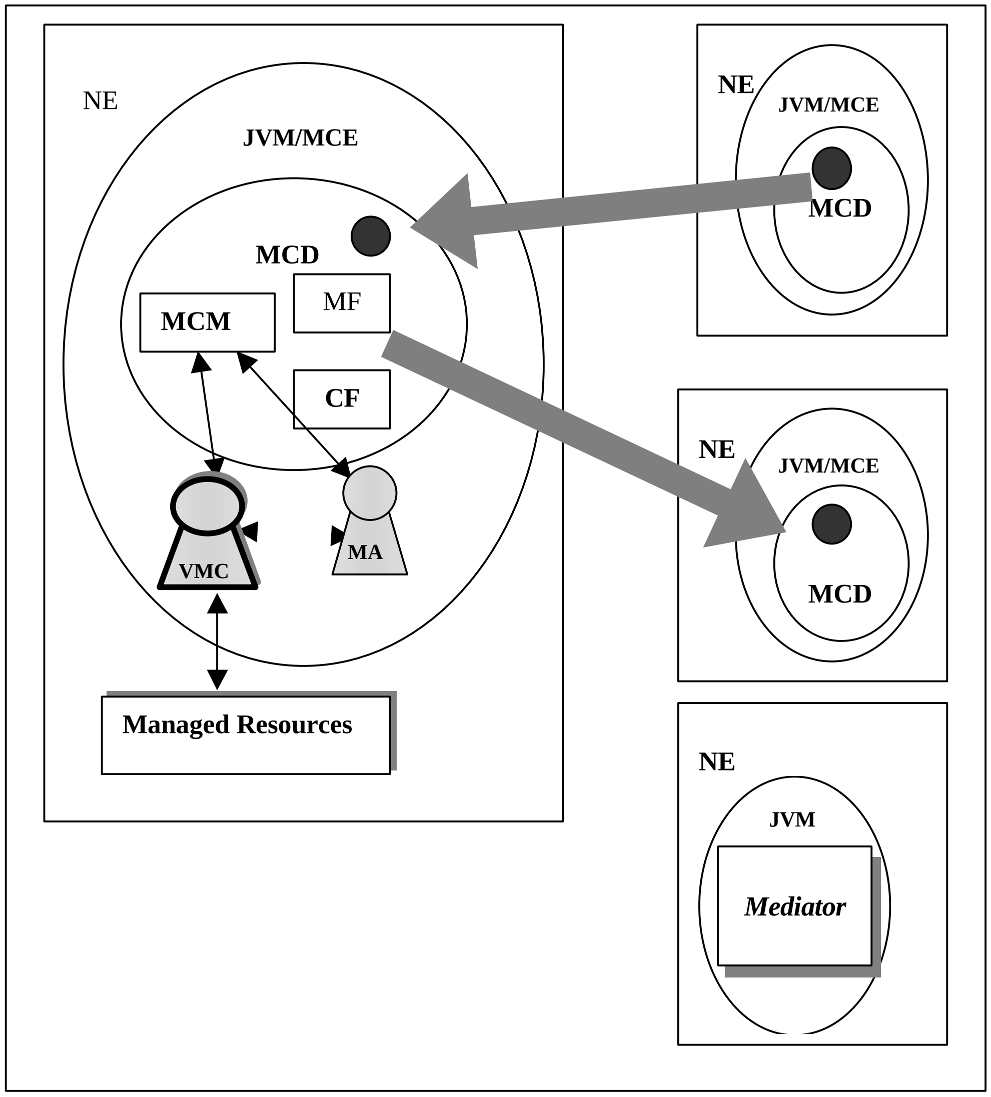


Figure 3.8 Mobile Code Environment Infrastructure

During the registration phase, the MCM announces the mobile agent's instantiation event, updates its migration history, if applicable, and depending upon whether or not the mobile agent has a persistent state, the MCM *restores* or *initializes* the mobile agent.

Once instantiated, the MCM *starts* the mobile agent and during the mobile agent’s lifetime, the MCM leads it to re-*start* its execution, to *suspend* temporarily and to *resume* executing, to *migrate* to another destination, or to *stop* and *destroy* the mobile agent. The MCM monitors the mobile agent during *communication* phases, sending or receiving messages to or from other mobile agents. Appendix A shows the chart diagram of the mobile agent’s execution status in the MCE.

The MCM provides an event-monitored mobile agent's execution status transition model: an *event dispatcher* announces to external observers (*event listeners*) the changes in the mobile agent's life cycle. The *eventDispatcher* monitors the changes in the mobile agent status and broadcasts them to all interested parties registered as *event listeners.*

3.3.2 Mobile Agent Location Services

Because mobile agents can travel around an MCD network, when a mobile agent wants to communicate with another mobile agent, the CF needs some kind of mechanism to ***locate*** the destination, either local or remote.

A mobile agent database and the MCM's event-monitored mobile agent's execution status transition model are the basis of the CF location mechanism. The database is a directory identifying mobile agents, their location, execution status and other attributes as well.

The following subsections describe how both, the local and the remote location mechanisms are implemented in the MCT.

3.3.2.1 The Local Mobile Agent Location Directory

The CF installs a directory, known as ***residents***, of all mobile agents visiting the local MCD. ***residents***, is registered with the *eventDispatcher* as an *event listener* in such a way that all the resident mobile agent's life cycle events announced by the *eventDispatcher* are caught by ***residents*** that adds, removes or updates the mobile agent status in the database accordingly.

3.3.2.2 The Regional Mobile Agent Location Directory

The MCT supports the concept of a region as a set of MCDs running within separated JVMs on the top of network elements (NEs). These MCDs subscribe to a regional mobile agent directory known as the ***Mediator***.

Mediator is the remote version of the ***residents***. It maintains an up-to-the-minute list of mobile agents running around the MCDs integrated in the region. Actually *remote event listeners* installed in each participating MCD maintain the ***Mediator*** directory. Again, in a MCD, a *remote event listener* associated with the ***Mediator*** catches the events related to the mobile agent's life cycle. Then, it communicates these changes to the ***Mediator*** using the RMI protocol.

residents and ***Mediator*** directories provide the MCEs with the capability to locate mobile agents either locally or region-wide respectively, not only for inter-mobile agent communication purposes but also for mobile agent management.

3.3.3 The Migration History

Migration is the ability of an mobile agent to stop its execution, save its state, and transport itself from a MCD to another in the network to continue executing in the new environment. Mobile agents can follow the default migration path established by all participating MCDs in the network. Alternatively, they can implement the migration patterns of their own or ultimately can select its migration path heuristically.

The approach of a network management system (NMS) based on mobile agent, in contrast with the traditional architecture followed by the centralized manager-agent model used by standards like SNMP, is to distribute the management functions through the delegation of specific management tasks (Management by Delegation: MbD) into mobile agents that are launched to the network as close the managed resources as possible to process the management data, taking advantage of local communication. Consequently, this NMS can be seen as a set of cooperative and task-oriented mobile management applications.

To illustrate, let us consider the *network model discovery* process using mobile agents. Discovery *netlets*[7] or *deglets*[8] can travel the network and dynamically create partial and specialized network models. The discovery selection criteria to model the network is application-oriented, in this case, management application-oriented. It can vary from a simple set of constrains to partition the network according to type of devices to encode more complex intelligent algorithms to manage certain states, behavior, negotiate services, etc. For instance, to diagnose network faults, the network manager can deploy a number of cooperative *netlets* that permanently navigate the network to selectively discover network devices exceeding various threshold values. Some of them, the active *netlets*, have been equipped to act autonomously and perform, if possible, certain actions on the faulty network devices.

Another example involving a cooperative number of mobile agents is to provide plug-and-play capabilities for configuring network components, a number of *netlets*

[7] A *netlet* in the context of the MCT, is a persistent mobile agent that migrates between network elements and executes on each of them; it is thought to never terminate. [4]

[8] A *deglet* in the context of the MCT, is a mobile agent sent to a remote location with a certain task to perform; the name comes from delegation of authority; it terminates after achieving its goal. [4]

or *deglets* can be launched to discover the network devices that will need the just installed component drivers.

Although a migration strategy exists to govern the mobile agent’s migration path, it may be advisable to record the actual mobile agent’s migration history. The migration history can be used to reinforce the migration strategy, adjusting it dynamically, or to determine when a *deglet* might stop executing. It can be used with management purposes to prevent or detect network performance problems that have arisen from the mobile agent's migration activity. The migration history can be used to prevent and/or detect the number of visited nodes exceeding a threshold value, multiple visits to the same network device, how far the *netlet* has gone from the limits of a given network domain, and so forth. That means the mobile agent's migration history can be used as part of the selection criteria to dynamically construct and maintain network models with management purposes.

In the MCT the mobile agent's migration history is defined as a list of MCD location identifiers sorted in the order by which they are visited by the mobile agent and is maintained in the mobile agent’s MobileCodeContainer. On migration the mobile agent, the migration history is also serialized and shipped to the next MCD destination.

The MCM is responsible for updating the mobile agent’s migration history. When a mobile agent is being registered by the MCM, it gets the mobile agent's container from the ***StorageManager***, and if it is the home MCD, and the mobile agent implements the *Migratable* interface, the migration history is initialized as an empty list. In any case, if *Migratable*, the current MCD location identifier is added to the mobile agent's migration history.

3.4 Summary

Autonomy, intelligence, mobility, and cooperation are capabilities distinguishing software agents as an adequate technology to be employed in the solution of complex distributed applications. OMG MASIF and FIPA standards are addressing interoperability issues between heterogeneous agent systems. Software agents also inter-operate with other software development technologies and more, software agents inter-operate with software systems.

The next chapter describes how the MCT has been equipped with a dual VMC implementation to integrate the mobile agent framework with the extensible *XMS-SNMP agent*. These two VMCs provide MCT’s mobile agents with capabilities to dynamically access and extend the SNMP's MIB.

Chapter 4 Integrating Mobile Agents with SNMP

4.1 Introduction

This chapter discusses how the *Mobile Code Toolkit* (MCT) is currently equipped with specialized SNMP management services and by the virtue of that, how mobile agents can perform management tasks taking advantages of their migration capabilities and of the local communication.

The basic idea for the integration of the mobile agent technology with traditional network management systems based on protocols like SNMP, is to distribute the management functionality spatial and temporarily throughout the network: that is, send the code closer to the spatial and temporarily distributed managed resources when needed. The organization of the *Mobile Code Toolkit - XMS-SNMP agent* integrated management platform handles both the spatial and temporal distribution of management services and the spatial and temporal distribution of managed resources using mobile agents.

With the solution provided in this thesis, mobile agents can be dynamically sent to the network element where the *XMS-SNMP agent* resides, have access to its MIB variables and accomplish their management functions. They can be sent to the targeted network element to extend the resident *XMS-SNMP agent*'s MIB as well.

Section 4.2 describes the design and implementation through which MCT provides the mobile agents the SNMP MIB access and extension services. Section 4.3 presents the complete MCT - *XMS-SNMP agent* integration environment.

4.2 Integrating the Mobile Code Toolkit with the XMS- SNMP agent

Performing their network management tasks, resident mobile agents might need access to local resources on a network element (NE) and for this purpose the MCT enables the Virtual Managed Component (VMC) as the standard interface through which resident mobile agents can gain access to managed resources on the NE.

In the MCE infrastructure resident mobile agents do not communicate directly with the extensible SNMP agent instead, mobile agents interface an intermediary VMC that encapsulates and provides an abstraction of the underlying communication protocol implementation and governs the mobile agent – extensible SNMP agent interaction. Therefore, it is conceivable that multiple VMCs could be active concurrently providing separate MCT – SNMP agent communication protocols on the same MCE.

Moreover, from the *XMS-SNMP agent*'s perspective, there are two types of interaction between the MCE and the *XMS-SNMP agent*:

I. Interaction derived from mobile agents acting in the role of subagents (mobile subagents) and dynamically extending the *XMS-SNMP agent*'s MIB, and

II. Interaction derived from mobile agents requesting data from the *XMS-SNMP agent*'s MIB.

Interaction derived from mobile subagents comprises three categories of messages.

I.1. Control messages:

- From mobile subagents to open, maintain and close connection with the SNMP agent;
- From mobile subagents to register/unregister a MIB sub-tree with the SNMP agent;
- From SNMP agent to unregister a MIB sub-tree;
- From SNMP agents to close connection with mobile subagents;

I.2. Data messages:

- From SNMP agent forwarding requests on behalf manager applications to mobile subagents for accessing the extended MIB;

I.3. Trap messages:

- From mobile subagents sending traps to manager applications.

As indicated in Section 2.3.2.1, the *XMS-SNMP agent* supports MIB extension capabilities via its DPI interface. Clearly, mobile subagents residing in the same network element as the extensible SNMP agent, would employ the local communication services provided by an intermediary VMC that hides the DPI protocol implementation details.

Interaction derived from mobile agents requesting data from the SNMP MIB includes not only messages requesting access to built-in SNMP MIB variables but also includes those messages requesting access to MIB variables implemented in mobile subagents.

The *XMS-SNMP agent* also accepts SNMP requests via its RDPI interface, usually from management applications running in the same network element. Taking advantages of the locality, mobile agents wanting to send management requests to the local *XMS-SNMP agent* would employ the RPDI communication services provided by an intermediary VMC that hides the protocol implementation details.

These two separate processes suggest that the integration between the MCT with the *XMS-SNMP agent* from the MCT's perspective be also governed by two independent VMCs interfacing the MIB extension and MIB access capabilities respectively.

Figure 4.1 shows the dual VMC architecture [36] embedded in the MCE running at the top of a Java Virtual Machine (MCE/JVM) and interfacing all MCE - *XMS-SNMP agent* communication. The *VMCMIBExtend* provides MIB extension capabilities to mobile agents wishing to act as mobile subagents of the extensible

SNMP agent. The other VMC, the *VMCMIBAccess*, provides MIB access capabilities to mobile agents wishing to retrieve MIB information from the extensible SNMP agent.

4.2.1 Mobile Agents extending the SNMP agent's MIB

This subsection discuses a *VMCMIBExtend* implementation called the ***MIBExtendFacilitator*** (MEF) component. This VMC provides MIB-extending mobile agents with two-way SNMP DPI communication capabilities between the mobile agent wishing to extend the MIB and the extensible SNMP agent.

This approach comes up with a *VMCMIBExtend* implementation as a facilitator, which can be set up at the MCD's startup time. At the time MIB-extending mobile agents are being instantiated and registered with the MCM in an MEF-enabled MCE, they are registered with the MEF.

When a MIB-extending mobile agent is registered with the MEF, as shown in Figure 4.2, the MEF associates the MIB-extending mobile agent with an individual DPI-enabled *MIBExtenderHandler* (MEH), running in a separate thread, to handle the two-way communication between the mobile agent and the also DPI-enabled extensible SNMP agent. At that point, the MEH is the one that actually opens the DPI connection with the SNMP agent and registers the mobile agent as a subagent[9]. Then, the MEF gets a reference of the MIB group list and attaches them to the SNMP agent's MIB.

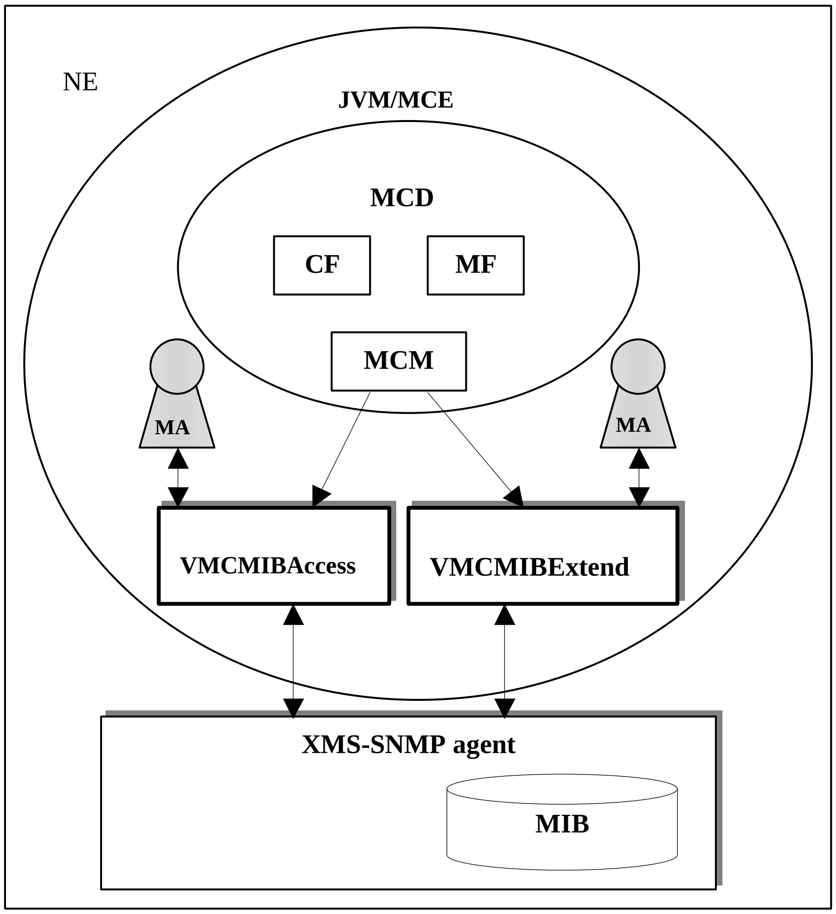


Figure 4.1 Dual VMC Architecture

Mobile agents wishing to act as an SNMP subagent extending the MIB must implement the *MIBExtender* interface. A *MIBExtender* mobile agent is also a VMC running in a MCD environment, implementing one or more MIB groups and dynamically registering them with the extensible SNMP agent.

[9] The *XMS-SNMP agent* also implements a management-MIB to maintain subagents and their dynamically registered MIB groups in the SNMP framework.

The *MIBExtender* interface provides functionality to process the **get**, **getNext**, and **set** SNMP management operations, to **unregister** a MIB group and to **close** the connection with the master agent on a manager request basis. The so-called *MIBExtender Engine* is mainly the *MIBExtender* interface implementation.

Once registered, the SNMP agent is capable of recognizing those requests for these dynamically registered objects and forwards the requests to the appropriated subagent using its devoted DPI open connection. Eventually, it will receive the responses and will send them to the management station. This scheme is illustrated in Figure 4.2.

4.2.2 Mobile Agents accessing the SNMP agent's MIB

The ***MIBAccessFacilitator*** component is a *VMCMIBAccess* interface implementation.

Requests coming from mobile agents are processed through a separate blocking RDPI communication link with the RPDI-enabled SNMP agent. Figure 4.3 shows what a request process looks like:

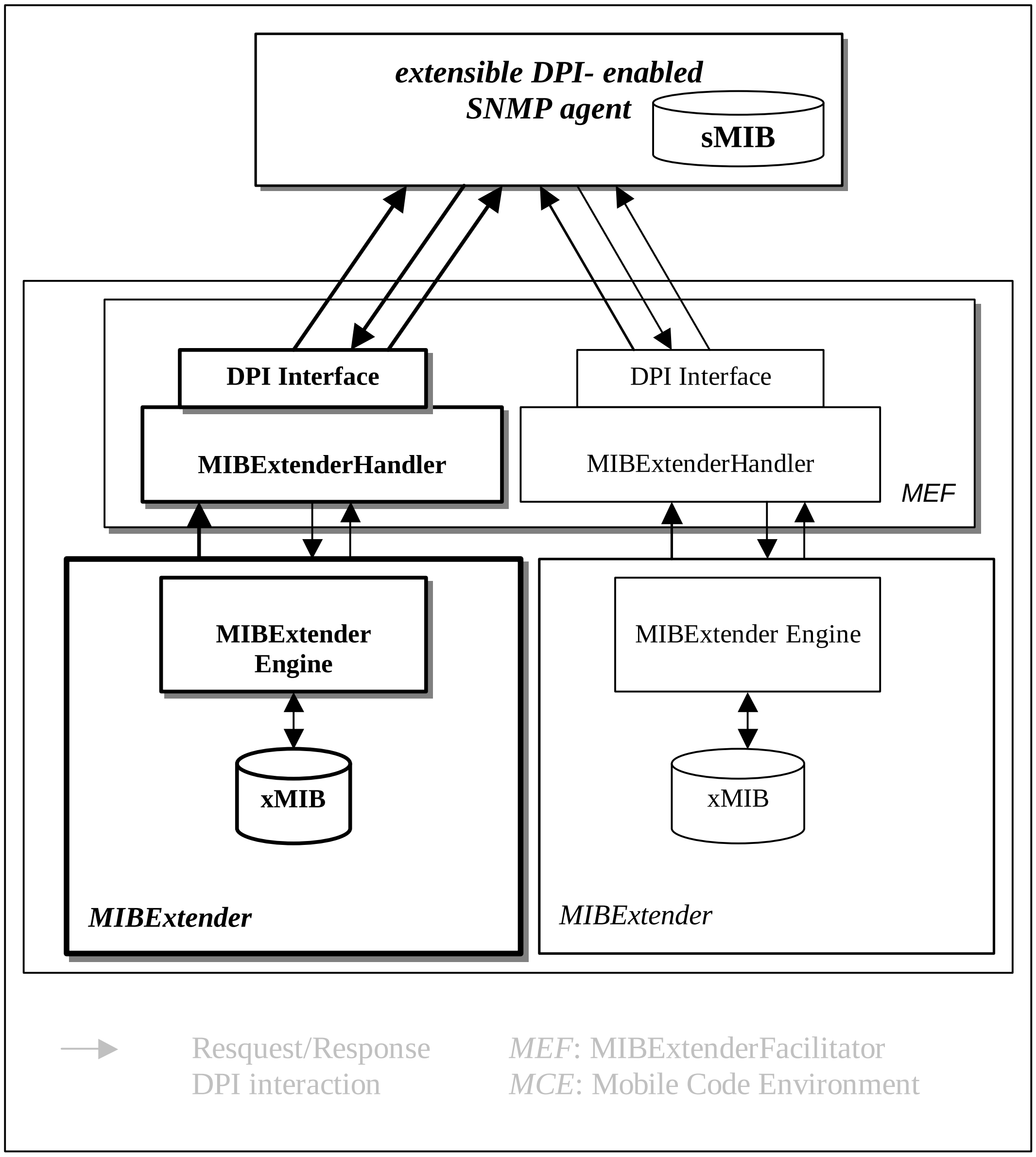


Figure 4.2 MIBExtender/MIBExtenderHandler Framework

```
opResponse  op ( opRequest, community){
    try {
              set up a RDPI connection with the SNMP agent;
              process the opRequest and create the RDPI packet;
              send the RDPI packet to the SNMP agent;
              block waiting for the response;
              strip away the RDPI elements from the results;
              return the results to the caller;
    } catch Exception () { …}

}
```

Figure 4.3 MIBAccessFacilitator Request Processing

4.3 The MCT – XMS-SNMP agent integrating environment

Figure 4.4 shows the complete integrated environment architecture. The network element (NE) contains the two components: the *XMS-SNMP agent* and the MCE running within a separate Java Virtual Machine (MCE/JVM).

The *XMS-SNMP agent* is listening for **get**/**getNext**/**set** requests from remote SNMP manager applications and thus, processes them and sends the replies back via the SNMP interface. Similarly, the agent is listening for requests from mobile agents running on the same network element (NE) and wishing to access the *XMS-SNMP agent*'s MIB; it is capable of processing them and sends the replies back through the RDPI interface.

Although the *XMS-SNMP agent* is listening for **open**/**register**/**unregister**/**close** DPI requests from mobile agents wishing to extend the MIB, it also originates (forwards) **get**/**getNext**/**set** requests for MIB extending mobile agents using the DPI interface.

The MCE/JVM provides SNMP network management capabilities. It configures the *MIBAccessFacilitator* (*MAF*) and the *MIBExtendFacilitator* (*MEF*) for MIB access and MIB extension capabilities respectively.

MIBExtenderHandlers (MEH) handle the two-way DPI communication between the associated MIB extender and the *XMS-SNMP agent*. Mobile agents wishing to access the *XMS-SNMP agent*'s MIB send requests to the *MAF*.

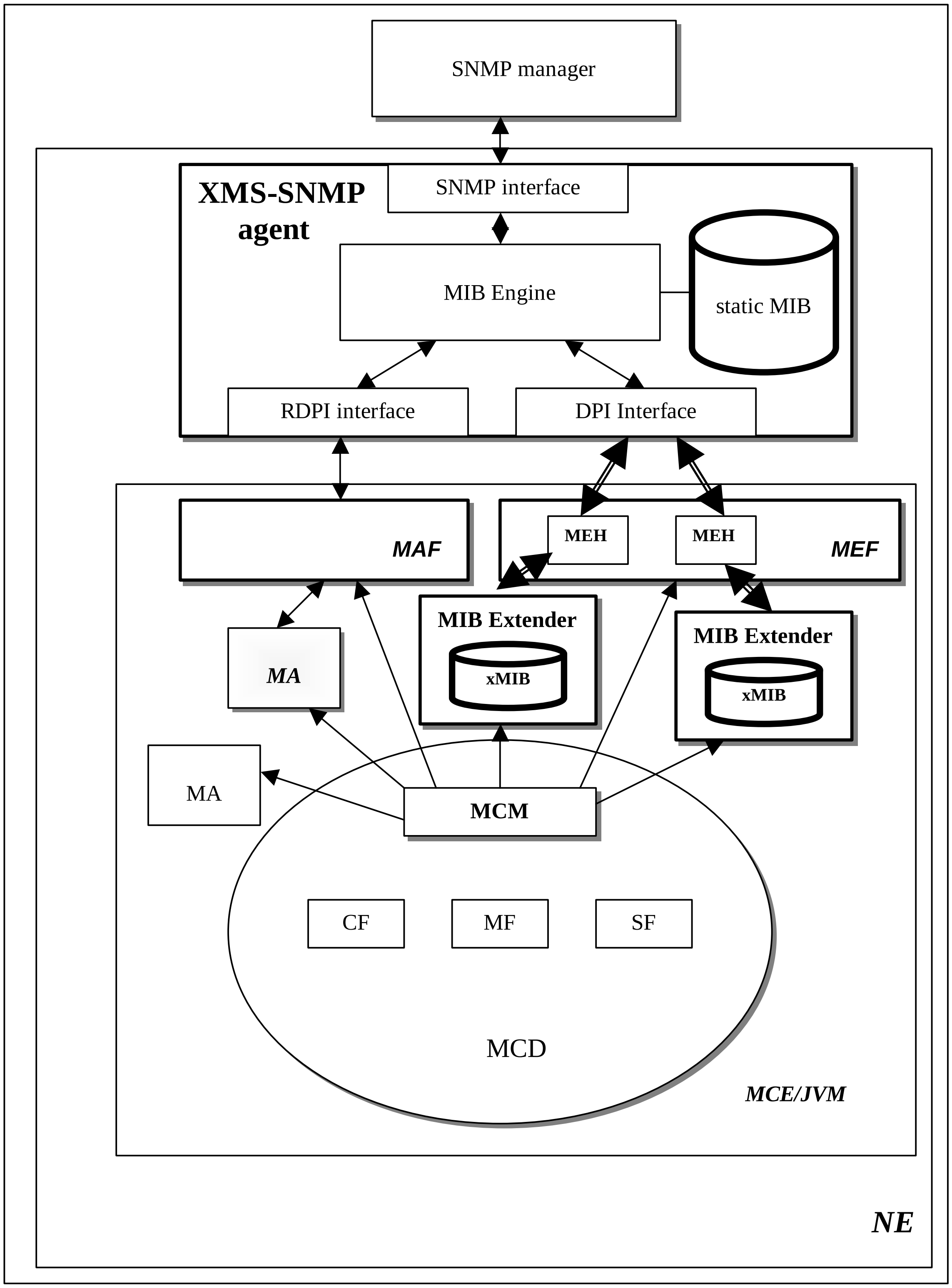


Figure 4.4 The MCT – XMS-SNMP agent Integrated Framework

4.4 Summary

The MCT - *XMS-SNMP agent* integrated environment is a framework for network management applications based on mobile agent technology. For the above management infrastructure to be feasible and complete in providing the needed services to network managers, mobile agent management supports should be provided.

The next chapter proposes a mobile agent management solution in the MCT - *XMS-SNMP agent* integrated environment.

Chapter 5 Mobile Agent MIB

5.1 Introduction

This chapter describes the design and implementation of a Management Information Base (MIB) for managing mobile agents running in a *Mobile Code Toolkit - XMS-SNMP agent* integrated management environment.

Doing their management tasks, mobile agents co-operate with each others and with the SNMP agent, access managed resources and, depending on the situation, they may decide to visit other NEs. Therefore, they are also network resources with specific tasks and network manager may want to control and monitor mobile agents as well.

In order to provide mobile agent management capabilities to network managers in a MCT- *XMS-SNMP agent* integrated environment, this thesis defines the structure and contents of a mobile agent MIB for providing the network manager applications the management functions of mobile agent location, execution status report, and control functions to apply to targeted mobile agents.

The *Mobile Code Toolkit* - *XMS-SNMP agent* infrastructure suggests that for managing mobile agents, a *MIBExtender* implementation interfaces the local mobile agents visiting the network element and extends the resident *XMS-SNMP agent*'s MIB to include the mobile agent MIB using the DPI interface provided by the *MIBExtendFacilitator*.

Section 5.2 describes the structure and contents of the mobile agent MIB and section 5.3 its implementation.

5.2 The Mobile Agent Management MIB

This section describes the structure and contents of the Mobile Code[10] Management Information Base (MCMIB) to monitor the mobile agents running around the network in an *XMS-SNMP agent*- MCT integrated environment.

The MCMIB is defined into a MIB group, known as *MCMIBTree* and comprises 2 MIB subgroups:

- *MCMIB*: defines the number and the attributes of mobile agents visiting the system, and
- *MHMIB*: defines the mobile agent's migration history.

[10] The term mobile code is similar to mobile agent and is used in the context of the MCT implementation.

Figure 5.1 shows the current position of the *MCMIBTree* group in the OSI registration tree. The *xmsAgentMIB* [57] was originally defined to support the *XMS-SNMP agent* management MIB group to manage mobile agents registered as subagents (mobile subagents) of the *XMS-SNMP agent*.

The following subsections examine each of the *MCMIBTree* subgroups.

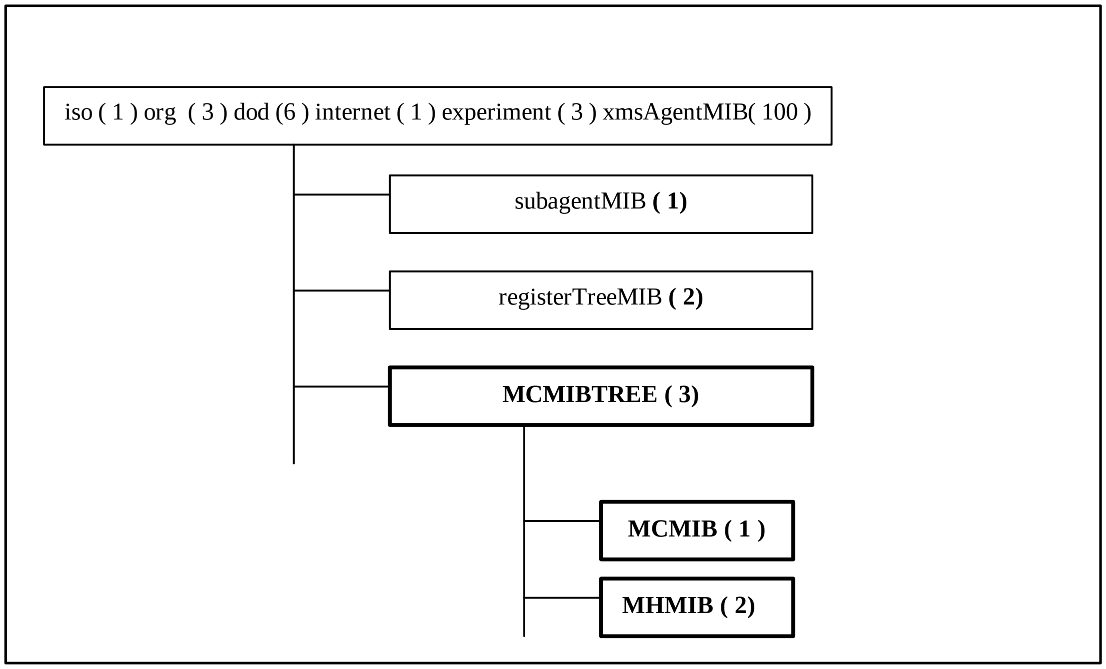

iso (1) org (3) dod (6) internet (1) experiment (3) xmsAgentMIB(100)
subagentMIB (1)
registerTreeMIB (2)
MCMIBTREE (3)
MCMIB (1)
MHMIB (2)

Figure 5.1 Position of the Mobile Agent MIB in the OSI Registration Tree

5.2.1 The MCMIB group

The MCMIB group provides general information (Figure 5.2) about the mobile agents instantiated in the system. The group includes the object type *mcNumber*, which represents the total number of mobile agents resident on the system. In other words, all the mobile agents instantiated in the system, independent of their current execution status.

The group also consists of the *mcTable*. This is a tabular object containing information about the visiting mobile agents. Each entry in the table has a *mcIndex* object, the *mcTable* index, whose value is an integer in the range of 1 and the current value of *mcNumber*; it uniquely identifies a mobile agent in the table.

The object *mcId* represents the descriptive mobile agent identifier. Its default format is:

mobileCodeName[autority]@className

The class full pathname in an MCE, the one that the mobile agent instantiates, can be retrieved from the *mcClassName* object. The *mcAlias* object type provides the list of alias names of this mobile agent.

mcInfo is the additional information the mobile agent can supply to the community.

mcType typifies the mobile agent as is in the MCE.

mcInfo represents additional information the mobile agent can provide to the community.

mcLocation identifies the MobileCodeDaemon where this mobile code is running. Its format is

hostaddress:tcpPort:udpPort:facPort

where

hostaddress is either the host IP address or its full domain name.

tcpPort is the TCP port the daemon is listening for incoming requests.

udpPort is the UDP port the daemon is listening for incoming requests.

facPort is the TCP port the daemon's communication facilitator(CF) uses.

The location uniquely identifies a daemon in a region

mcStatus represents the current mobile agent’s execution status as defined in the MCE. Appendix A shows the mobile agent's execution status chart diagram.

The *mcMessagingAccess* value indicates whether the mobile agent has inter-mobile agent communication capabilities.

The *mcMigratable* object provides information about the migration capabilities the mobile agent has, and the value of the *mcVisitedNodes* object indicates the number of MCDs this mobile agent has visited up to the minute. If the mobile agent has no migration capabilities[11], the value of this object will be zero.

mcMgmtOp is the only object in the *mcTable* entry that has read-write access permissions. The purpose of this object is to allow a network management station to control the mobile agent's execution status. This object provides capability to the manager to *start*, *suspend*, *resume*, *stop*, and *destroy* a mobile agent registered in the system.

5.2.2 The MHMIB group

The *MHMIB* group provides the information related to the mobile agent's migration history.

The MIB group consists of only one tabular object, the *mcMigrationHistoryTable*.

As shown bellow in Figure 5.3, the *mcMigrationHistoryTable* is double indexed by *mcIndex* and then by *mhIndex*. The *mcIndex* value matches that of *mcIndex* for one of the entries in the *mcTable* in the MCMIBGroup.

[11] This mobile agent has no mobility capability at all. Actually it is stationary.

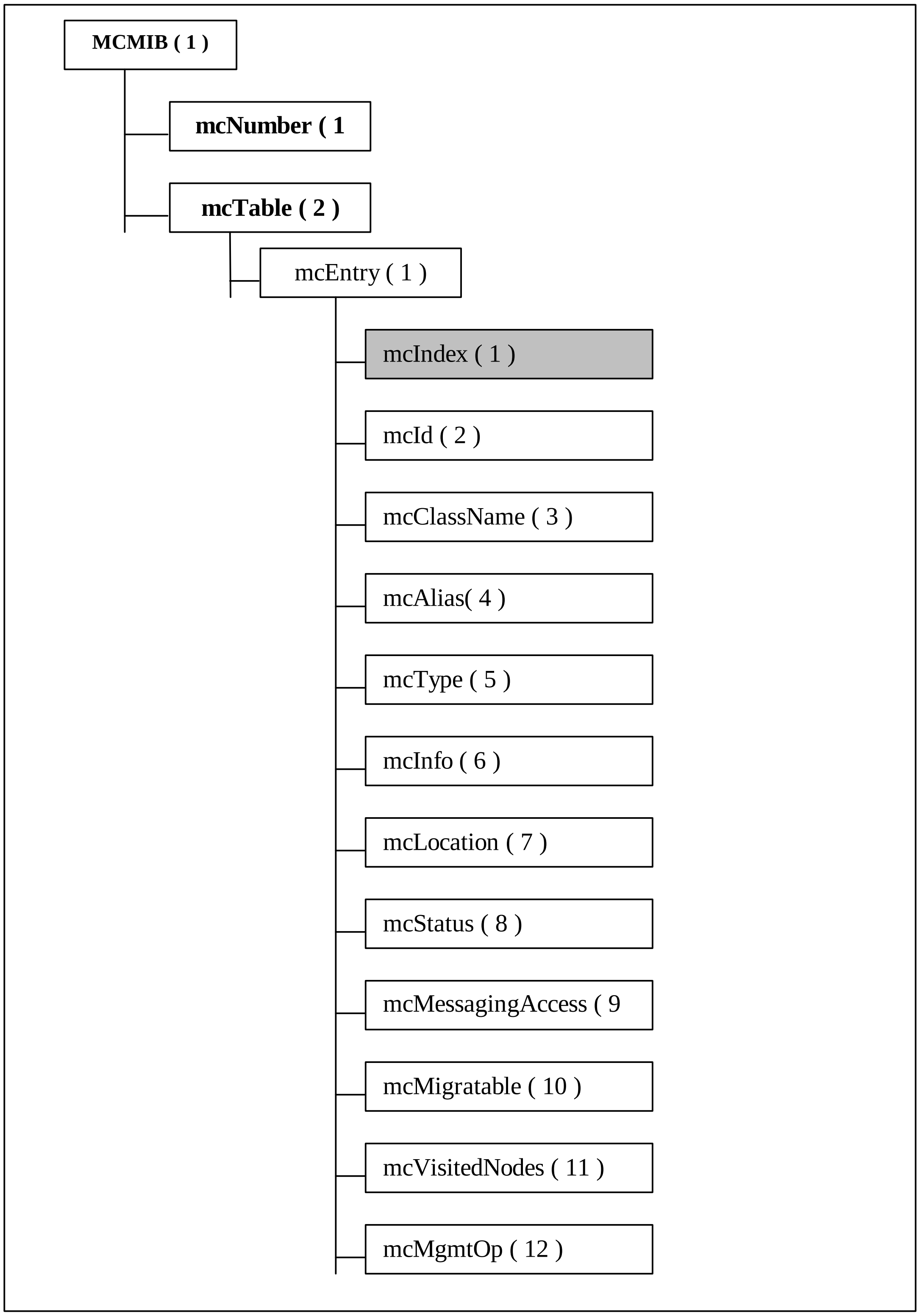


Figure 5.2 Mobile Code MIB Group

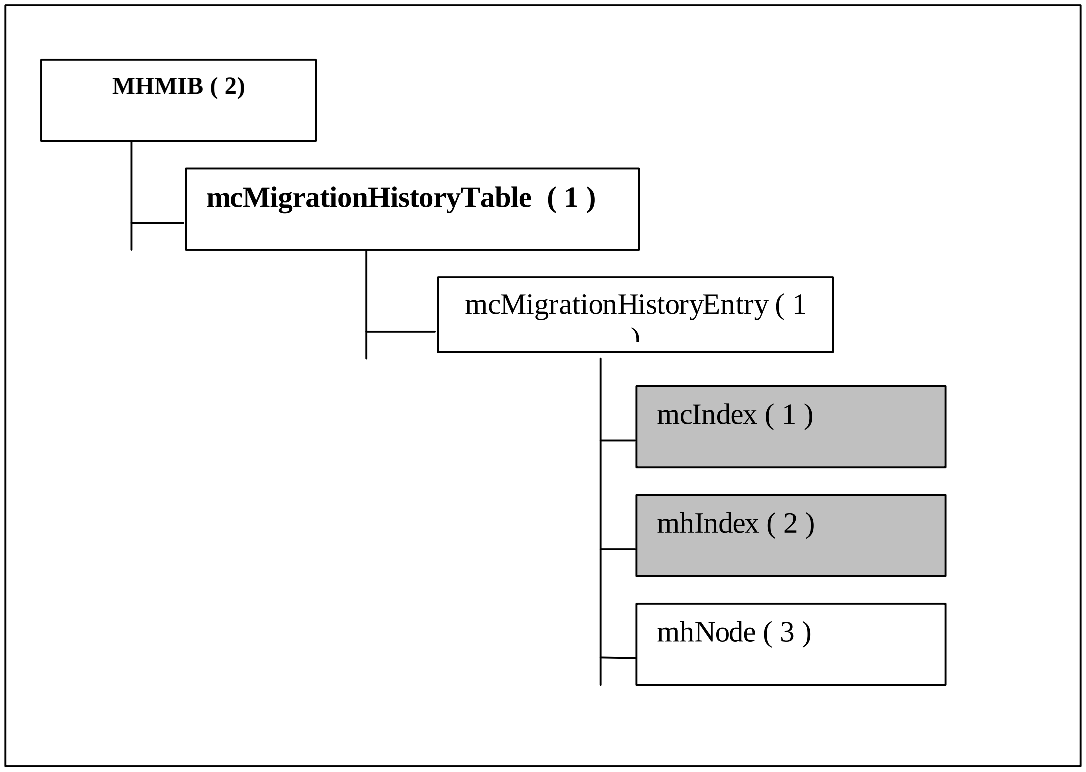


Figure 5.3 Migration History MIB Group

The table contains an entry for each MCD visited by a migratable mobile agent. The value of the *mcVisitedNodes* in *mcTable* also indicates the number of rows in the *mhMigrationHistoryTable* referring to the same mobile agent.

A migratable mobile agent will always have a least one entry in the *mcMigrationHistoryTable* and the first of them is, of course, its home MCD.

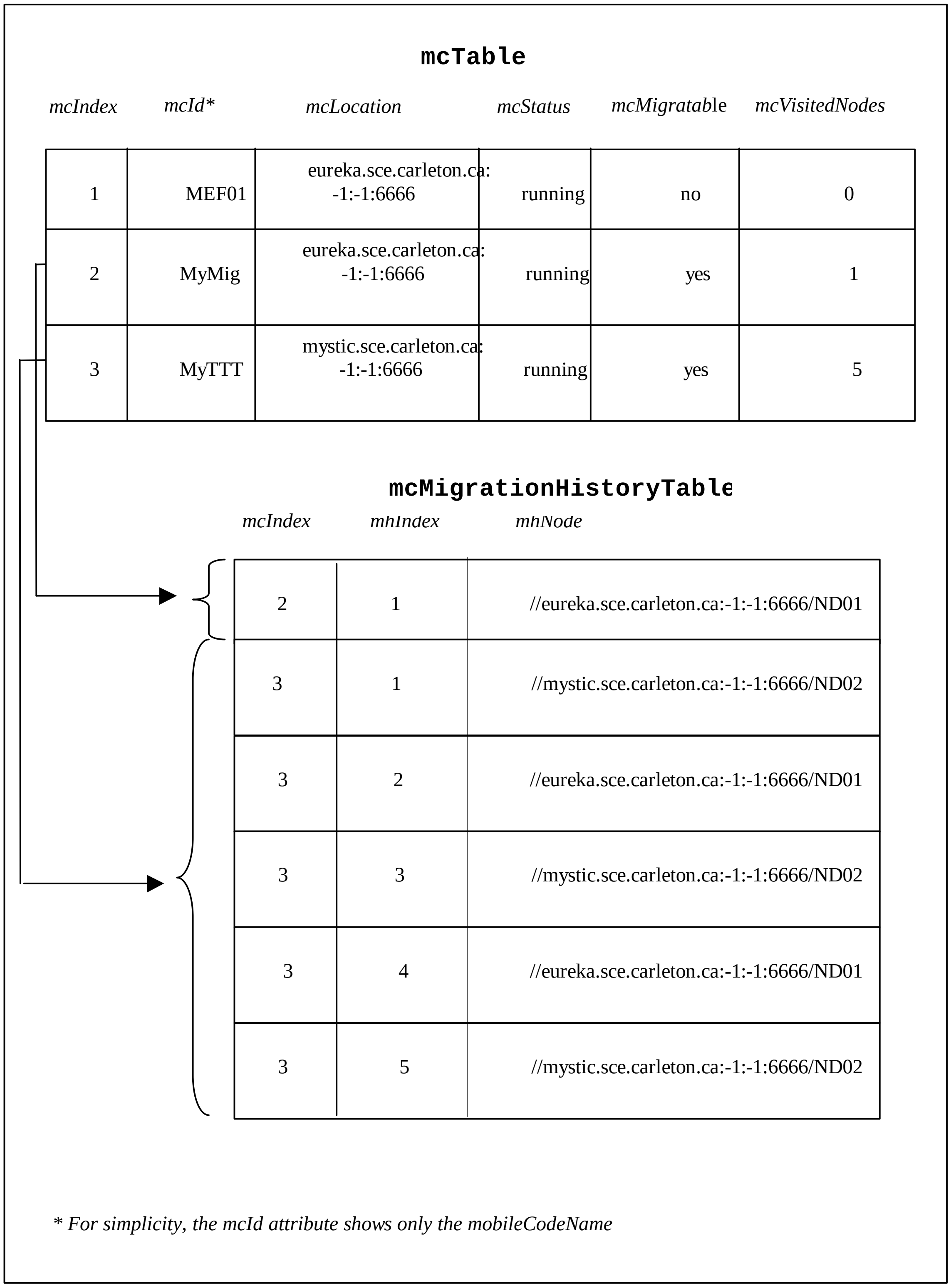
mhNode specifies the location of the visited mobile agent system. It is a string with the format:

//hostadress:tcpPort:udpPort:facPort/nodeId

Figure 5.4 is an example of a specific instance view of these 2 MIB groups that illustrates the relationship between the 2 tables. In this case, the mobile agent table has three entries. The first entry corresponds to a non-migratable mobile agent, but the second and third correspond to migratable ones. Therefore, these two last entries have one or more related entries in the migration history table.

For example, the second entry of MEF01 corresponds to a migratable mobile agent, created, instantiated, and now running in MCD identified by *ND01*; and the value of *mcVisitedNodes* of 1 indicates MEF01 has not migrated from its home MCD.

The third entry of MyTTT shows this mobile agent has visited 5 MCDs. The MCDs and the order they have been visited can be found in the migration history table.



mcTable

mcIndex	*mcId**	*mcLocation*	*mcStatus*	*mcMigratable*	*mcVisitedNodes*
1	MEF01	eureka.sce.carleton.ca:-1:-1:6666	running	no	0
2	MyMig	eureka.sce.carleton.ca:-1:-1:6666	running	yes	1
3	MyTTT	mystic.sce.carleton.ca:-1:-1:6666	running	yes	5

mcMigrationHistoryTable

mcIndex	*mhIndex*	*mhNode*
2	1	//eureka.sce.carleton.ca:-1:-1:6666/ND01
3	1	//mystic.sce.carleton.ca:-1:-1:6666/ND02
3	2	//eureka.sce.carleton.ca:-1:-1:6666/ND01
3	3	//mystic.sce.carleton.ca:-1:-1:6666/ND02
3	4	//eureka.sce.carleton.ca:-1:-1:6666/ND01
3	5	//mystic.sce.carleton.ca:-1:-1:6666/ND02

** For simplicity, the mcId attribute shows only the mobileCodeName*

Figure 5.4 Instance Views of mcTable and mcMigrationHistoryTable

5.3 The Mobile Agent MIB Implementation

Subsection 5.3.1 presents the ***MCMIBExtender,*** a *MIBExtender* interface implementation supported by a stationary mobile agent; subsection 5.3.2 presents the *MCMIB* and *MHMIB* group implementations and discusses how the mobile agent MIB objects can be mapped to the actual MIB variables.

5.3.1 The MCMIBExtender

MCMIBExtender stands for Mobile Code MIB Extender. As indicated, it is a *MIBExtender* implementation, whose main function is to interface with the mobile agents with the purpose of SNMP management, implementing the two MIB groups, *MCMIB* and *MHMIB* presented in section 5.2. The implementation chosen is a *stationary* agent that can be installed at the MCD's startup time.

Figure 5.5 depicts the ***MCMIBExtender*** framework in the context of a NE environment where the ***MIBExtendFacilitator*** has been installed in the MCE and an *XMS-SNMP agent* local version is running.

When installed and registered by the MCM, ***MCMIBExtender*** is registered with the ***MIBExtendFacilitator***. At this moment, the DPI-enabled MEH is already associated with the *MCMIBExtender*, opens the DPI connection with the *XMS-SNMP agent* and registers the *MCMIBExtender* as a subagent of the *XMS-SNMP* *agent*. The MEH enters in a forever loop, waiting for requests coming from the master agent.

The MEH forwards the request to the *MCMIBExtender* and, in turn, accesses the MIB variables in the two MIB groups accordingly. *MCMIBGroup* and *MHMIBGroup* implement the MIBGroup abstract class, which contains the data and behaviour of a MIB group.

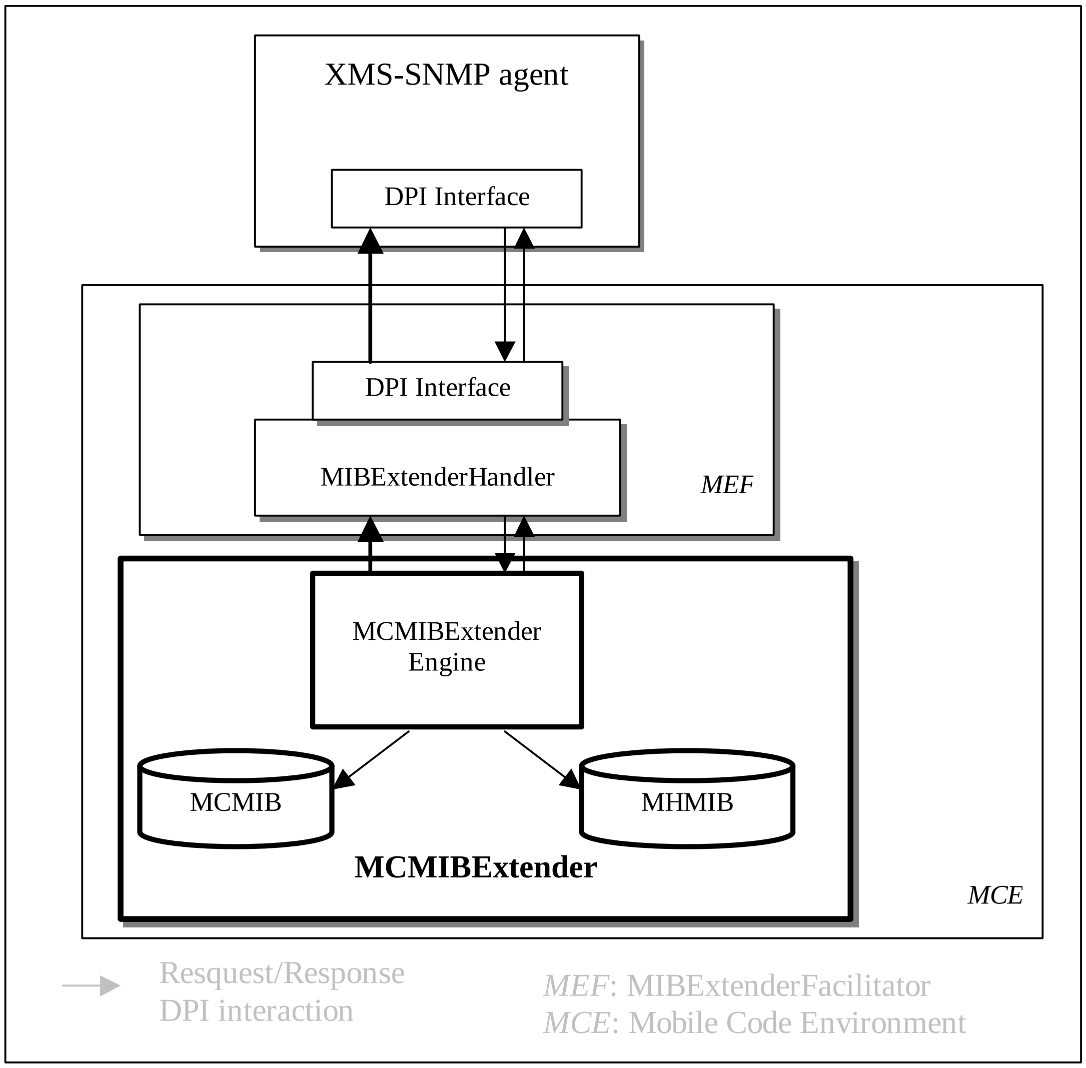


Figure 5.5 MCMIBExtender Framework

Which mobile agents the *MCMIBExtender* and its two MIB groups are really interfacing is discussed next.

5.3.1.1 Local Mode Management

The first approach implements a *MCMIBExtender* interfacing with local mobile agents, that is, the mobile agents running on the top of the local MCD. Therefore, a network management system based on the extensible *XMS-SNMP agent* and the MCT technology might be configured in such a way that each network element having an MCE must install the MEF and its local *MCMIBExtender* and run the *XMS-SNMP agent*.

Figure 5.6 displays an example of an integrated MCT- SNMP network management system. The region consists of three Java-enabled network elements (NE) and a network management station (NMS). Each NE is running an *XMS-SNMP agent* and has created an MCE where an MCD is running on the top of a separated JVM. *ND01*, *ND02* and *ND03* identify the MCDs, respectively. Each MCE has a single *MCMIBExtender* interfacing with the local mobile agents running in the MCE.

SNMP management requests are sent to the specific NE's *XMS-SNMP agent*.

The problem with this approach is that those NEs participating in the management system must have sufficient resources in its operating environment to run the *XMS-SNMP agent* and a separated JVM for an MCE as described above.

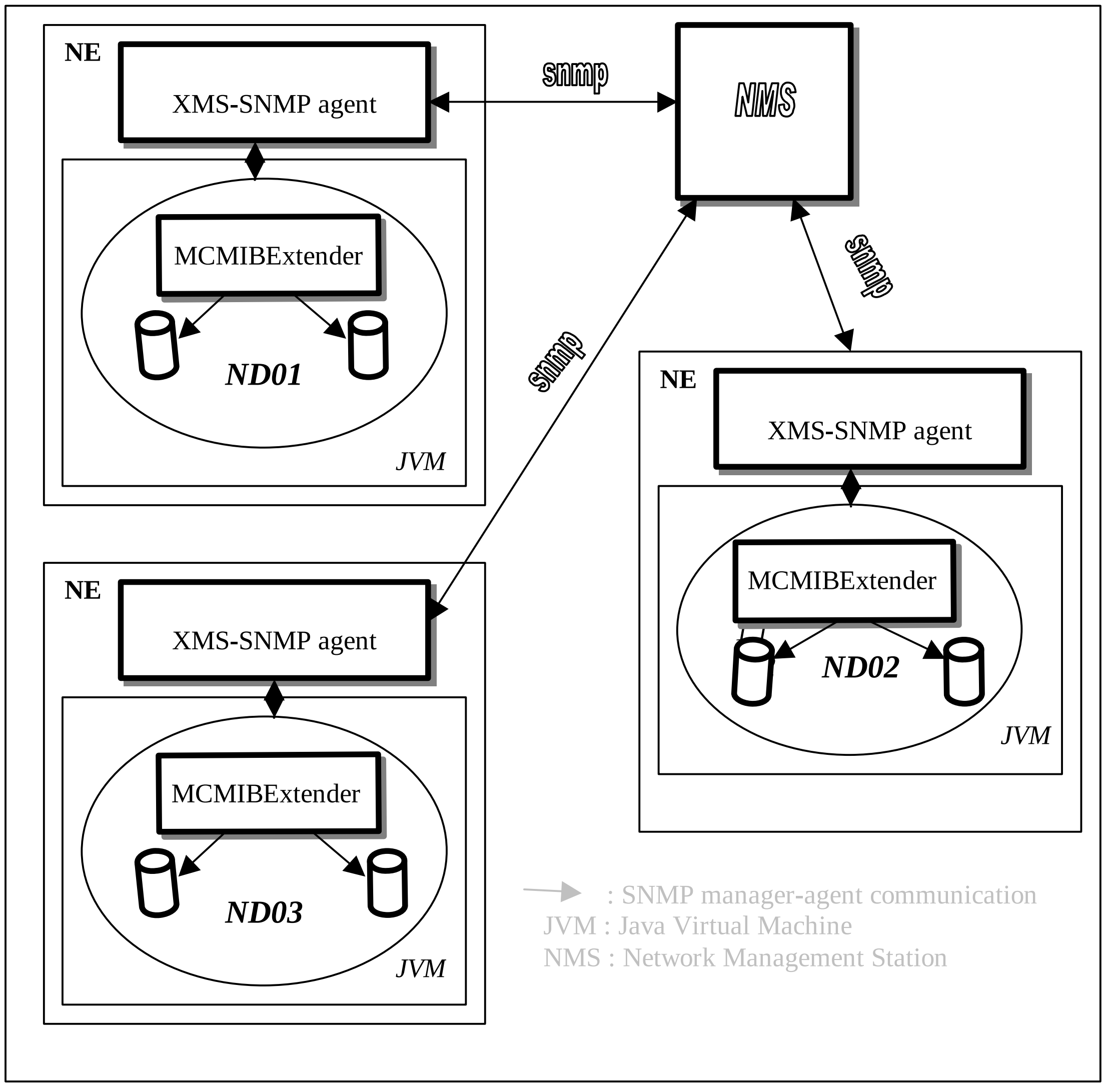


Figure 5.6 Mobile Agent Management in Local Mode

5.3.1.2 Regional Mode Management

To overcome these difficulties, a second approach uses the MCT's region-wide location service. In this case, only one of the NEs running its MCE must instantiate the *MCMIBExtender* and run the extensible *XMS-SNMP agent*. A single mobile agent location directory, the *Mediator*, will be set up in a separated JVM and the *MCMIBExtender* interfaces with the mobile agents registered in this regional *Mediator*.

Figure 5.7 displays a configuration example where the regional mode management approach is chosen.

There are three MCEs, *ND01*, *ND02*, and *ND03*, and the single *Mediator*, each running in a separated JVM. At the time of their startup, the MCDs identify the *Mediator* as the region-wide location directory. Only *ND01* is configured to register the *MCMIBExtender* as a subagent of the local *XMS-SNMP agent*. Therefore, the *ND01*'s *MCMIBExtender* is encouraged to interface with all mobile agents running around all over the region.

Currently a MCMIBExtender can be set up to interface mobile agents running on the top of a each MCD, or a single MCMIBExtender interfaces all mobile agents running around a community of MCDs integrated in a region.

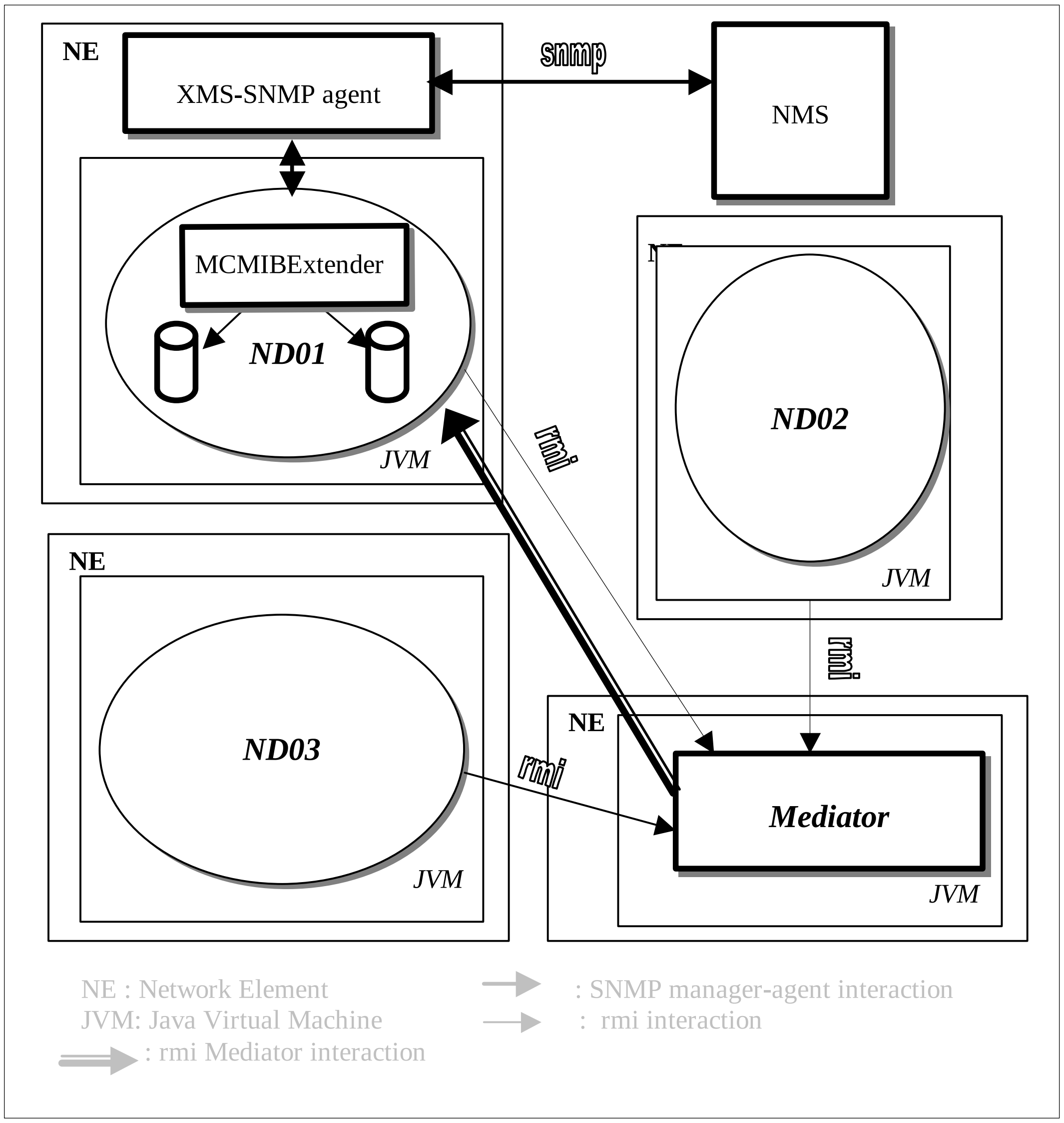


Figure 5.7 Mobile Agent Management in Regional Mode

5.3.2 The MCMIBGroup and the MHMIBGroup

The actual data and behavior of a MIB group is encapsulated in the MIBGroup abstract class. *MCMIBGroup* and *MHMIBGroup* both are MIBGroup implementations that provide the access functions to the actual mobile agent MIB variables.

During the initialization phase, *MCMIBExtender* instantiates both *MCMIBGroup* and *MHMIBGroup*, specifying the management mode: local or regional. If local management, the local mobile agent's location directory, called *LocalMCDirectory*, essentially provides the actual MIB variables that map into the MCMIBTree definition. In regional management, however, the MIB variables will be retrieved directly from the regional mobile agent location directory, known as the *Mediator*.

To access the actual MIB variables, *MCMIBGroup* and *MHMIBGroup* first get a reference to the local or regional location directory using the **getDirectory**() function. The access function retrieves the directory reference from the MCM.

The MIB object *mcNumber* in *MCMIBGroup* is a scalar object and has no correspondent variable in the location directories. Instead, the access function **mcNumber()** is provided and computes the object value by counting the number of mobile agents currently registered in the just-referenced directory.

The MIB objects in the *mcTable* in *MCMIBGroup* are tabular objects where each row in the MIB definition file corresponds to a specific mobile agent in the location directory. *mcIndex* is the MIB object which uniquely identifies a particular mobile agent in the table. The access function **getMobileCodeRecord**() uses the value of the *mcIndex* to locate the corresponding *MobileCodeRecord* (MCR) in the referenced directory.

As soon as the MCR is obtained, the attributes can be readily retrieved from the access function **getAttributeValue**(). Attributes *mcId* (2), *mcClassName* (3), *mcAlias* (4), *mcType* (5), *mcInfo* (6), *mcLocation* (7), *mcStatus* (8), *mcMessagingAccess* (9), *mcMigratable* (10), correspond directly with MCR's id, className, alias, info, location, status, hasMessagingAccess and migratable attributes respectively; *mcVisitedNodes* (11) can be retrieved from the MCR's history attribute size.

When processing a SNMP set request involving a MIB variable registered by the subagent, the *XMS-SNMP agent* – DPI subagent interactions consist of a DPI set-commit-action packet chain sent from the SNMP agent and received by the subagent. When a set request chain for a *mcMgmtOp* (12) MIB object is received, the command access function **action**() is the one that does the real management operation. It gets a reference to the corresponding management interface (*LocalManager* or *RemoteManager*) and uses it to issue the management operation itself.

As it is shown in Figure 5.3, the MIB objects in the *mcMigrationHistoryTable* in the MHMIBGroup are tabular objects and *mcIndex* and *mhIndex* identify a row in the table. For a management request, the access function **getHistoryDataAt**() will first locate the *mcIndex*-th mobile agent in the location directory and, secondly, will retrieve the *mhIndex*-th element in its MCR's *history* field.

Appendix C shows an example of the interaction among components during a SNMP management request process in a regional mode management scenario.

5.4 Summary

This chapter has focused on mobile agent management in a *Mobile Code Toolkit* - *XMS-SNMP agent* integrated environment. It presented the design and implementation of a mobile agent management MIB that can be used in an SNMP-based network management system to control mobile agents running around a MCT network. Appendix B provides the formal managed object definitions for the mobile agent MIB.

The *MCMIBExtender* is the actual mobile agent MIB extender that holds the attributes and status of mobile agents and responds to the requests from the *XMS-SNMP agent* in order to report the execution status of the mobile agents and apply management commands for starting, suspending, resuming, stopping and destroying targeted mobile agents.

Chapter 6 Test Design

6.1 Introduction

This chapter focuses on the development of test strategies and cases to demonstrate the efficiency and the robustness of the solutions presented in this research work.

The *Mobile Code Toolkit – XMS-SNMP agent* integrated framework enables a network management model based on mobile agent, distributing management functions through the delegation of specific management tasks into mobile agents that are launched to the network as close the managed objects as possible and interact with the resident SNMP agent. The current implementation uses the DPI and Reverse DPI protocols in order to provide mobile management applications with SNMP management capabilities and to enhance the efficiency of this mobile agent – SNMP agent interaction. The RDPI protocol is used instead of the SNMP protocol by mobile management applications arriving to the same network element wanting to access the *XMS-SNMP agent*'s MIB variables. Although both protocols are analytically compared in [36] and [55], it also important to measure how much faster the RDPI protocol is to demonstrate the benefits of using it in a local communication scenario.

For the purpose of managing mobile agents, the *Mobile Code Toolkit* has been extended with a mobile agent MIB that can interface mobile agents visiting the local MCE. One of the advantages of the dual VMC architecture providing separate SNMP management capabilities to mobile agents is that if there is no resident *XMS-SNMP agent*, it is still possible to configure the MCEs in such a way the mobile agent MIB interfaces all mobile agents visiting a region.

Section 6.2 describes the test strategy and cases employed to measure the RDPI and the SNMP protocols' response times. Section 6.3 provides the test strategy and cases to demonstrate how network management application can manage mobile agents in a region domain employing the MCT – *XMS-SNMP agent* integrated environment.

6.2 RDPI and SNMP Protocol Comparison Test

This test measures the response times out the RDPI and SNMP requests sent to the extensible SNMP agent, being both the *XMS-SNMP agent* and the management applications executing in the same network workstation.

6.2.1 Test Strategy

6.2.1.1 Test Configuration

Figure 6.1 shows the test configuration. The *XMS-SNMP agent*, the RDPI and SNMP management tools are all resident in *eureka*.

eureka is a 75 MHz Sun SPARCstation 5, running the SunOS 5.7 operating system and 128 MB of main memory. The *XMS-SNMP agent* is a modified version of UCD-SNMP agent, written in C and compiled on SunOS 5.7. The *XMS-SNMP agent* receives SNMP, DPI and RDPI messages from the non well-known ports 1061, 1064 and 1065 respectively.

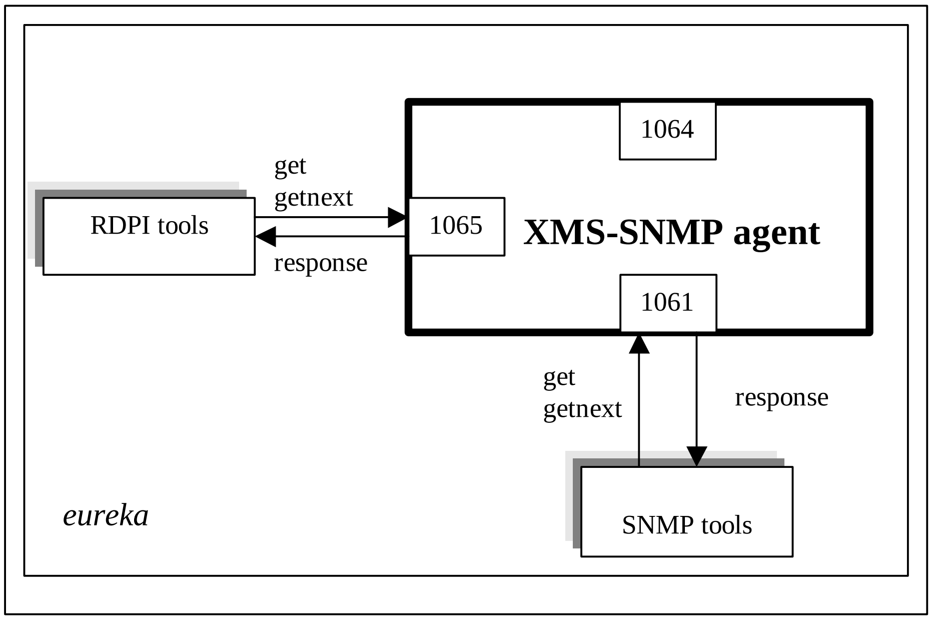


Figure 6.1   Test Configuration

6.2.1.2 The RDPI Manager tools

The ***RDPIGet*** is a modified command line application written in Java that measure the time elapsed between a RDPI request packet is created, sent to the *XMS-SNMP agent* and the corresponding response packet is received.

6.2.1.3 The SNMP Manager tool.

The University of Quebec at Montreal (UQAM)'s management tools are a set of graphical applications written in Java and its Requester has been modified to measure the time elapsed between a SNMP request packet is being BER encoded, sent to the *XMS-SNMP agent* and the corresponding response packet have been received and decoded.

6.2.2 Test Scenario

In order to obtain more consistent results, the number of replications and the length of the request's variable binding list the management tools sent to the *XMS-SNMP agent* are determined at run-time. Timestamps were taken in every replication, each time recording the response time and finally calculating the average. Both tools calculate and print out the *average response time* as result.

6.2.3 Test cases

The management tools are executed repeatedly, polling the same MIB variables over the same number of replications and sending their respective final results to a text file.

The *XMS-SNMP agent* is polled repeatedly alternating between the RDPI and the SNMP management tools 5 times each (10 repetitions), over 100 and 500 replications, requesting the value of 1 and 3 MIB variables respectively.

Test cases are summarized in Table 6.1.

Test Case	1	2	3	4
Replications	100	500	100	500
Var Bind List Length	1	1	3	3

Table 6.1. Test Cases

The experiment was conducted and the results are shown in Table 6.2. The RDPI request average response time is, in all these test cases, approximately 2-3 times faster than the SNMP's.

Test Case	1	2	3	4
SNMP	15.42	14.06	16.22	15.79
RDPI	**5.91**	**4.84**	**7.67**	**6.52**

Table 6.2. Average Response Times (msecs)

These tables show two important results. First, the RDPI protocol not only has a message overhead (meaning the extra octets needed for encoding and decoding purposes) smaller than the SNMP's but also the *XMS-SNMP agent* can decode an RDPI message faster than an SNMP message [55]. Second, the bigger the size of the *varbind* list involved in a management operation the smaller the performance overhead required for the RDPI protocol to process requests.

6.3 Mobile Agent MIB Test

This section demonstrates how a network manager can manage mobile agents running in an *Mobile Code Toolkit - XMS-SNMP agent* integrated environment.

6.3.1 Test Strategy

6.3.1.1 Test Configuration

For this purpose, the chosen network management system comprises a region of three Java-enabled networks management stations, *sunspot.sce.carleton.ca*, *inm-057178.sce.carleton.ca* and *inm-057179.sce.carleton.ca*. Figure 6.2 displays the network management system configuration: *sunspot* is a SunOS Release 5.7 workstation running an *XMS-SNMP agent* daemon and there is a MCE running a MCD on a separated JVM, identified as *ND01*. *inm-057178* and *inm*-057179 are Windows NT 4.0 workstations running *ND02* and *ND03* MCDs respectively. These MCDs register with the *Mediator*, the regional database, which is also running on a separated JVM on *inm-057178*.

6.3.2 Test Scenarios

There are two management scenarios, the local mode management scenario where the *MCMIBExtender* configured on *ND01*, interfaces the mobile agents visiting the local host. The regional mode management scenario is similar to the previous one but the *MCMIBExtender* interfaces all mobile agents running around the registered region.

6.3.3 Test Cases

6.3.3.1 Local Mode Management Test Case

In the local mode management scenario, the network manager application queries the *XMS-SNMP agent* and will be able to manage the mobile agents visiting the local MCE. In this case, the *ND01* is configured with a *MIBExtendFacilitator* and a *MCMIBExtender*, which extends the *XMS-SNMP agent*'s MIB, interfacing the mobile agents visiting *sunspot*'s MCE.

Appendix D shows the complete sequence for this test case.

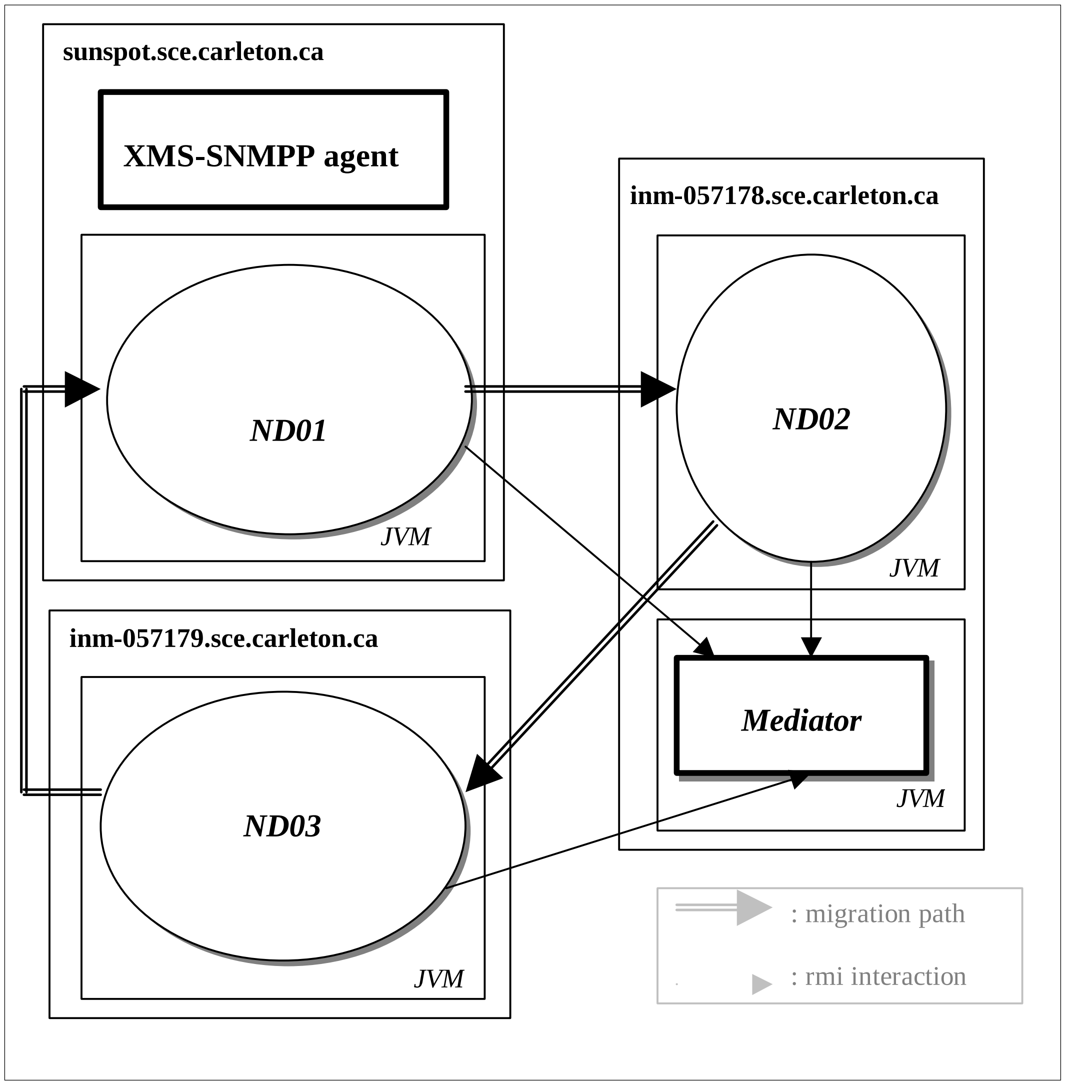


Figure 6.2 Network Management System Configuration

6.3.3.2 Regional Mode Management Test Case

In a regional management scenario, the network manager station queries the *XMS-SNMP agent* and will be able to manage all mobile agents running in the region. In this case, the *ND01* is configured with a *MIBExtendFacilitator* and a *MCMIBExtender*, which extends the *XMS-SNMP agent*'s MIB, interfacing the mobile agents visiting the three MCDs currently registered with the Mediator at *sunspot*, *inm-057178* and *inm-057179* respectively. It also requires the three netlet daemons configure the *RemoteManager* component to accept management operations coming from the remote network manager application.

Appendix E shows the complete sequence for this test case.

6.4 Summary

The Mobile Agent MIB has been designed for managing mobile agents residing in a MCT network and provides network managers with information about their identity, classification, current location, operational status and their migration history.

From the implementation point of view, the MCE MIB extension facilitator is provided in the way of a stationary *VirtualManagedComponent*, known as the *MIBExtendFacilitator*. On the other side, the Mobile Agent MIB is implemented as another stationary mobile agent, the *MIBExtender* that can be configured local or regional. If regional, the *MCMIBExtender* interfaces all mobile agents registered in the *Mediator* and the network manager can issue management operations to start, suspend, resume, stop and destroy mobile agents all over the region.

Chapter 7 Conclusions

7.1 Thesis Summary

The basic idea of the mobile agent – based network management systems is to distribute the management functionality throughout the network, to manage rapid changes and the scalability of today's complex and heterogeneous networks.

This research work has focused on the integration of the mobile agent technology with the SNMP network management framework. It has extended the **Perpetuum Mobile Procura** Project's *Mobile Code Toolkit* with SNMP management capabilities to integrate with the extensible *XMS-SNMP agent*. Mobile agents arriving at the same network element of the SNMP agent can request SNMP MIB variables using the RDPI protocol as the means that governs the local communication between mobile agents and the SNMP agent. Mobile agents arriving at the same network elements of the SNMP agent can extend the SNMP MIB using a DPI-enabled communication channel that handle all the communication between the MIB extender and the SNMP agent.

The efficiency of using the Reverse Distributed Protocol Interface (RDPI) in order to enhance the mobile agent – SNMP agent interaction is also demonstrated. The average response times of management operations using RDPI and SNMP requests sent to the *XMS-SNMP agent* in a local environment are measured, and the results indicated that RDPI is about three times faster than SNMP.

Finally, the result is also the design and implementation of a Management Information Base (MIB) to allow network management applications to monitor, control and act on mobile agent behavior in a *Mobile Code Toolkit - XMS-SNMP agent* integrated environment. The Mobile Agent MIB provides network management applications with general information about the mobile agents; it holds information like identity, class, type, current location, execution state, and the migration history of mobile agents. The MIB also provides network managers applications with control functions for starting, suspending, resuming, stopping and destroying targeted mobile agents.

7.2 Future Work

During this research work, a number of directions have been identified for future work. So far, only a limited information about mobile agents is provided but, particularly in the provisioning of management support based on mobile agents, it would be desirable to develop a mobile agent management model which allow managers to administrate the way and order mobile agents need and access resources of the network elements they are visiting. Therefore managers can dynamically influence in the life cycle of mobile agents anticipating and avoiding or reducing significantly problems like security violations, congestion, deadlock, system breakdown, and etceteras.

Other research direction might include the extension of management capabilities not only to mobile agents but also to Mobile Code Environments participating in a regional domain. The management system supports monitoring and controlling the list of agents residing in a particular MCE, adaptive configuration of environment preferences, monitoring and control of NE's resources utilization.

References

[1] Abeck, S., Köppel, A., Seitz, J., *A Management Architecture for Multi-Agent Systems*, Proceedings of the IEEE Third International Workshop on System Management, 1998. Pages: 133-138.

[2] Apache XML Project, *Xerces Java Parser*, [ON LINE] URL: http://xml.apache.org/xerces-j/index.html.

[3] Baldi, M., Picco, G.P., *Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications*, Proceeding of the IEEE 1998 International Conference on Software Engineering, 1998.Pages: 146-155.

[4] Bieszczad, A. , Pagurek, B., *Network Management Application-Oriented Taxonomy of Mobile Code*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium, NOMS 98, IEEE, Volume: 2, 1998. Pages: 659-669 [ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html.

[5] Bieszczad, A., Pagurek, B., *Towards Plug-and-Play Networks with Mobile Code*, Proceedings of the International Conference for Computer Communications ICCC'97, November 19-21, 1997.
[ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html.

[6] Bieszczad, A.,White, T., Pagurek, B, *Mobile Agents for Network Management*, IEEE Communication Surveys, September, 1998.
[ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html

[7] Case, J., McCloghrie, K., Waldbusser, S., *Manager-To-Manager Management Information Base*, RFC 1451, April 1993.

[8] Finin, T., Labrou, Y., Peng, Y., *Mobile Agents Can Benefit from Standards Efforts on Interagent Communication*, IEEE Communications Magazine, July 1998. Volume: 36, Issue: 7, Pages: 50-56.

[9] FIPA, *FIPA'97 Specifications,*
[ON LINE] URL: http://www.fipa.org/spec/fipa97.html.

[10] FIPA, *FIPA'98 Specifications,*
[ON LINE] URL: http://www.fipa.org/spec/fipa98.html.

[11] FIPA, *Foundation for Intelligent Physical Agents,*
[ON LINE] URL: http://www.fipa.org/

[12] Goldszmidt, G., Yemini, Y., *Delegated Agents for Network Management*, IEEE Communication Magazine, Volume: 36, Issue: 3, March 1998.

[13] Goldszmidt, G., Yemini, Y., *Distributed Management by Delegation*, Proceedings of the 15th International Conference on Distributed Computing Systems, 1995. Pages: 333-340.

[14] Green Shaw, Hurst L., Nangle, B., Cunningham, P., Somers, F., Evans, R., *Software Agents: A review,*
[ON LINE] URL: http://www.cs.tcd.ie/research_groups/aig/iag/toplevel2.html

[15] Gschwind, T., Feridun, M., Pleisch, S., *ADK-Building Mobile Agents for Networks and Systems Management from Reusable Components*, Proceedings of the First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents., 1998.

[16] IBM, *Aglets Specification 1.1 Draft* ,

[ON LINE] URL: http://www.trl.ibm.co.jp/aglets/spec11.html

[17] IETF, *DIStributed MANanagement Working Group*,

[ON LINE] URL: http://www.ietf.org/html.charters/disman-charter.html.

[18] IKV++, *Grasshopper, The Agent Platform*,

[ON LINE] URL: http://www.ikv.de/products/grasshopper/.

[19] Jasmin Project, A Script MIB implementation.

[ON LINE] URL: http://www.ibr.cs.tu-bs.de/projects/jasmin/.

[20] JIDM, *Joint Inter-Domain Management*, [ON LINE] URL: http://www.jidm.org/.

[21] Knight, G., Hazemi, R., *Mobile Agent based management in the INSERT Project*, Journal of Network and Systems Management, Vol. 7, No.3, 1999.

[22] Levi, D., *Introduction to the Script MIB*, The Simple Times, Vol. 7, Number 2, November 1999.

[ON LINE] URL: http://www.simple-times.org/pub/simple-times/issues/7-2.html#applications.

[23] Levi, D., Schönwälder, J., *Definition of Management Objects for Scheduling Management Operations*, RFC 2591, 1999.

[ON LINE] URL: http://www.ietf.org/rfc/rfc2591.txt?number=2591

[24] Levi, D., Schönwälder, J., *Definition of Management Objects for the Delegation of Management Scripts*, RFC 2592, 1999.

[ON LINE] URL: http://www.ietf.org/rfc/rfc2592.txt?number=2592

[25] Lopes, R.P., Oliveira, J.L., *On the Use of Mobility in Distributed Network Management*, Proceedings of the IEEE 33th Hawaii International Conference on System Sciences, 1998. Volume: Abstract, 2000. Pages: 47.

[26] Luchuck, A., *Schedule MIB*, The Simple Times, Vol. 7, Number 2, November 1999.

[ON LINE] URL: http://www.simple-times.org/pub/simple-times/issues/7-2.html#introduction.

[27] Luderer, G., Ku, H., Subbiah, B., Narayanan, *A., Network Management Agents Supported by a Java Environment*, Submitted to ISINM'97.

[28] Magedanz, T., *Agent Cluster Baseline Document*, version 0.2, Advanced Communications Technologies and Services.

[ON LINE] URL: http://olympus.algo.com.gr/acts/dolphin/AC-baseline.htm.

[29] McManus, É., *Script MIB Implementation Experience*, The Simple Times, Vol. 7, Number 2, November 1999.

[ON LINE] URL: http://www.simple-times.org/pub/simple-times/issues/7-2.html#introduction.

[30] Meyer, K., Erlinger, M., Betser, J., Sunshine, C., *Decentralizing Control and Intelligence in Network Management*, Proceedings of the Fourth International Symposium on Integrated Network Management, 1995.

[31] Nwana, H.S, *Software Agents: An Overview*, Cambridge University Press, 1996.

[ON LINE] URL:

http://www.sce.carleton.ca/netmanage/docs/AgentsOverview/ao.html.

[32] Nortel Networks, *FIPA-OS*.

[ON LINE] URL: http://www.nortelnetworks.com/fipa-os/.

[33] ObjectSpace, Voyager ORB, [ON LINE] URL: http//www.objectspace.com/.

[34] OMG, *Common Object Request Broker Architecture*,
[ON LINE] URL: http://www.corba.org/.

[35] OMG, *Object Management Group*,
[ON LINE] URL: http://www.omg.org/.

[36] OMG, *MASIF- RTF Results, orbos/98-03-09*.
[ON LINE] URL: ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf

[37] Pagurek, B., Wang, Y., White, T., *Integration of Mobile Agents with SNMP: How and Why*, Network Operations and Management Symposium, 2000, NOMS '00, IEEE/IFIP, 2000. Pages: 609-622.
[ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html

[38] Park, J., *Implementation of SNMP Agent by Inferno*.
[ON LINE] URL: http://www.lucent-inferno.com/Pages/Developers/Documentation/White_Papers/snmp.html.

[39] Pavlou, G., *Using Distributed Object Technologies in Telecommunication Network Management*, IEEE Journal on Selected Areas in Communications, Vol.18, No. 5, May 2000.

[40] *Perpetuum Mobile Procura Project*,
[ON LINE] URL: http://www.sce.carleton.ca/netmanage/perpetuum.shtml

[41] Pinheiro, R., Poylisher, A., Caldwell, H., *Mobile Agents for Aggregation Network Management Data*, Proceedings of the IEEE Third International Symposium on

Agent Systems and Applications and Third International Symposium on Mobile Agents, 1999. Pages: 130-140.

[42] PTS, *Voyager ORB*,
[ON LINE] URL: http://www.pts.com/.

[43] Schönwälder, J., *Network Management by Delegation from Research Prototypes Towards Standards*, Proceedings JENC8.

[44] Schramm, C., Bieszczad, A. and Pagurek, B., *Application-Oriented Network Modeling with Mobile Agents*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, February 1998. IEEE, Volume: 2, 1998. Pages: 696-700.

[45] Silva, L.M., Soares, G., Martins, P., Batista, V., Santos, L., *The Performance of Mobile Agent Platforms*, Proceedings of the IEEE First International Symposium on Agent Systems and Applications Third International Symposium on Mobile Agents, IEEE 1998. Pages: 270-271.

[46] Simões, P., Moura, L., Boavida, F., *Integrating SNMP into Mobile Agent Infrastructure*, DSOM, 1999.

[47] Simões, P, Reis, R., Silva, L.M., Boavida, F., *Enabling Mobile Agent Technology for Legacy Network Management Frameworks*, SoftComm'99.

[48] SNMP Research International, *EMANATE The Enhanced MANagement Agent Through Extensions*,
[ON LINE] URL: http://www.snmp.com/products/emanate.html.

[49] Staab, S., Erdman, M., Maedche, A., Decker, S., *An Extensible Approach for Modeling Ontologies in RDF(S)*.

[ON LINE] URL: http://www.aifb.uni-karlsruhe.de/WBS.

[50] Stallings, W., *SNMP, SNMPv2, SNMPv3, RMON 1 and 2*, Third Ed., Addison-Wesley, 1999.

[51] Strauß, F., Script MIB Performance Analysis, The Simple Times, Vol. 7, Number 2, November 1999.

[52] Sun Microsystems, *Java Management eXtensions*,
[ON LINE] URL: http://java.sun.com/products/JavaManagement/.

[53] Susilo, G., Bieszczad, A. and Pagurek, B., *Infrastructure for Advanced Network Management based on Mobile Code*, Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, 1998.
[ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html.

[54] UCD, *UCD-SNMP Agent*,
[ON LINE] URL: http://ucd-snmp.ucdavis.edu/.

[55] UMBC, *Specification of the KQML agent-communication language*
[ON LINE] URL: http://www.cs.umbc.edu/kqml/kqmlspec/spec.html.

[56] Waldbusser, S., *Remote Network Monitoring Management Information Base*, RFC 1271, November 1991.
[ON LINE] URL: http://www.ietf.org/rfc/rfc1271.txt?number=1271.

[57] Wang, Y., *Integration of Mobile Agent Environment with Legacy SNMP*, M.Eng. Thesis, Dept. Systems and Computer Engineering, Carleton University, Ottawa, Canada, August 1998.

[58] White T., Pagurek B., and Bieszczad A., *Network Modeling For Management Applications Using Intelligent Mobile Agents*, accepted for publication in a special

issue on Mobile Agents of the Journal of Network and Systems Management to be published in September, 1999.

[ON LINE] URL: http://www.sce.carleton.ca/netmanage/publications.html.

[59] Wijnen B., Carpenter G., Curran K., Sehgal A., Waters G., *The SNMP Distributed Protocol Interface*, RFC 1592, March 1994

[ON LINE] URL: http://www.ietf.org/rfc/rfc1592.txt?number=1592

[60] World Wide Web Consortium, *Resource Descriptor Framework (RDF) Schema Specification 1.0*,

[ON LINE] URL: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[61] World Wide Web Consortium, *SiRPAC – Simple RDF Parser and Compiler*,

[ON LINE] URL: http://www.w3.org/RDF/Implementations/SiRPAC/.

[62] Yemini, Y., Goldzmidt, G., Yemini, S., *Network Management by Delegation*, Proceeding of the Second International Symposium on Integrated Network Management, 1991.Pages: 95-107

Appendix A

Mobile Code's Execution Status Chart Diagram

Figure A.1 shows the mobile code's execution status chart diagram in the MCT environment.

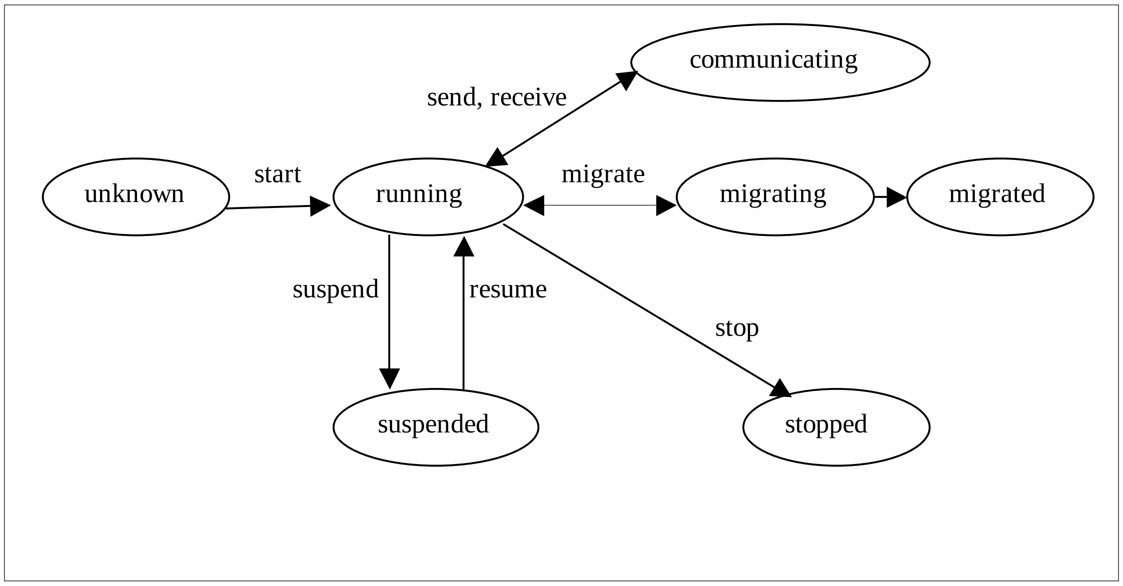


Figure A.1 Mobile Code Status Transition Graph

The MCM declares the mobile code's execution status as UNKOWN during the time elapsed between MCM instantiates the mobile code and issues the start command to put the mobile code to run. As soon as the mobile code is started, its status changes to *RUNNING*.

Meanwhile running, a mobile code can receive multiples suspend/resume command pairs and its status will switch between *SUSPENDED* and the *RUNNING* status. The MCM temporarily changes the mobile code's execution status if the mobile code is sending or receiving messages to *COMMUNICATING*. At the point in time when the communication finishes, its execution status returns back to *RUNNING*.

The MCM registers the mobile code as *MIGRATING* when the mobile code is to migrate. If the migration process is successful, the execution status will evolve to MIGRATED and eventually the mobile code will be destroyed. Only a Migratable mobile code can enter in the *MIGRATING* status; that means, a stationary one will never turn to this status.

The MCM can stop the mobile code execution. Once a mobile code is *STOPPED*, there is no way to return back the mobile code's execution status but destroy it.

Appendix B

Managed Objects Definitions for Mobile Code MIB

```
-- Mobile Code Management MIB Definition


MC-MIB DEFINITIONS  ::= BEGIN


IMPORTS


    OBJECT-TYPE
        FROM RFC1212
    IpAddress, TimeTicks
        FROM RFC1155
    DisplayString
        FROM RFC1213-MIB



xmsAgentMIB       OBJECT-IDENTIFIER  ::= {1 3 6 1 3 100}
-- subagentMIB     OBJECT-IDENTIFIER  ::= {xmsAgentMIB 1}
--regTreeMIB       OBJECT-IDENTIFIER::=  {xmsAgentMIB 2}



mcMIBTree          OBJECT-IDENTIFIER  ::=  {xmsAgentMIB 3}
```

```
mcMIB              OBJECT-IDENTIFIER  ::=  {mcMIBTree 1}

mhMIB              OBJECT-IDENTIFIER  ::=  {mcMIBTree 2}


-- mcMIBTree groups


--mcMIB group

-- This MIB group defines the number and the list of mobile codes entries

-- for  the mobile codes running in  the system.

-- The system scope could be a local mobile code system (a Mobile Code Daemon) or

-- a  mobile code region (a Mediator ).


mcNumber OBJECT-TYPE

          SYNTAX  INTEGER ( 0..255 )

          ACCESS   read-only

          STATUS   mandatory

          DESCRIPTION

            "The number of mobile code currently running in  the system"

     ::= {mcMIB 1}


mcTable OBJECT-TYPE

   SYNTAX SEQUENCE OF MCEntry

   ACCESS not-accesible

    STATUS mandatory
```

```
     DESCRIPTION
        " A list of mobile code entries. The number of entries is given
          by the value of   mcNumber"
    ::= {mcMIB 2}

**mcEntry** OBJECT-TYPE
    SYNTAX MCEntry
    ACCESS not-accesible
    STATUS mandatory
    DESCRIPTION
         "A mcEntry contains objects related to the attributes of mobile codes"
    INDEX {mcIndex}
    ::= {mcTable 1}

**MCEntry** ::= SEQUENCE {
 mcIndex
    INTEGER,
mcId
   DisplayString(SIZE(0..80)),
mcClassName
   DisplayString(SIZE(0..80)),
mcAlias
   DisplayString(SIZE(0..80)),
mcTyps
  DisplayString(SIZE(0..80)),
mcInfo
  DisplayString(SIZE(0..80)),
mcLocation
  DisplayString(SIZE(0..80)),
mcStatus
  INTEGER,
mcMessagingAccess
    INTEGER,
mcMigratable
    INTEGER,
mcVisitedNodes
  INTEGER,
mcMgmtOp
 INTEGER
}

mcIndex OBJECT-TYPE
     SYNTAX INTEGER
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
```

```
        "A unique value identifying the mobile code in the mcTable.
         It ranges between 0 and the value of mcNumber."
    ::= {mcEntry 1}
```

mcId OBJECT-TYPE

```
     SYNTAX DisplayString(SIZE(0..80))
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
        "The mobile code identifier, as is in a MCD.
          It is a unique value within the scope of the region.
          Its default format is name[authority]@className."
    ::= {mcEntry 2}
```

mcClassName OBJECT-TYPE

```
     SYNTAX DisplayString(SIZE(0..80))
     ACCESS read-only
     STATUS mandatory
     DESCRIPTION
        "The  fullpath class name this mobile code instantiates.
        For instance, mct.mcmgmt.MIBExtender ".
 ::= {mcEntry 3}
```

mcAlias OBJECT-TYPE

```
    SYNTAX DisplayString(SIZE(0..80))

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        "The list of alias names this mobile code is known in the MCTcontext".

 ::= {mcEntry 4}


mcType OBJECT-TYPE

    SYNTAX DisplayString(SIZE(0..80))

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        "The mobile code' type as defined in the MCT context.

         For instance, Mobile Code, VirtualManagedComponent, NETLET, EXLET, etc.".

 ::= {mcEntry 5}


mcInfo OBJECT-TYPE

    SYNTAX DisplayString(SIZE(0..80))

    ACCESS read-only

    STATUS mandatory

    DESCRIPTION

        "Additional information the mobile code can provide to the community".

 ::= {mcEntry 6}
```

```
mcLocation OBJECT-TYPE
    SYNTAX DisplayString(SIZE(0..80))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Identifies the MobileCodeDaemon where this mobile code is running.
         Its format is   hostaddress:tcpPort:udpPort:facPort
         hostaddress could be the host IP address or its full domain name.
         tcpPort is the TCP port the daemon is listening for incoming requests.
         udpPort is the UDP port the daemon is listening for incoming requests.
         facPort is the TCP port the daemon's communication facilitator(CF) uses.
         The location  uniquely identifies a daemon in a region".
 ::= {mcEntry 7}

mcStatus OBJECT-TYPE
    SYNTAX INTEGER {
          UNKNOWN (0),
          RUNNING (1),
          SUSPENDED (2),
         MIGRATING (3),
         MIGRATED (4),
         COMMUNICATING (5),
        STOPPED (6)
      }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "The  mobile code execution status in the MCD context".
 ::= {mcEntry 8}
```

mcMessagingAccess OBJECT-TYPE

```
    SYNTAX INTEGER {
         YES (0),
         NO (1)
      }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Whether or not the mobile code is a communicating
 mobile code  in  the MCD context".
 ::= {mcEntry 9}
```

mcMigratable OBJECT-TYPE

```
    SYNTAX INTEGER {
         YES (0),
         NO (1)
      }
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "Whether or not the mobile code is a migratable mobile code".
  ::= {mcEntry 10}
```

mcVisitedNodes OBJECT-TYPE

```
   SYNTAX INTEGER
   ACCESS read-only
   STATUS mandatory
   DESCRIPTION
       "Number of nodes this mobile code has visited during its life cycle.
        It should be zero for non-migratable mobile codes and it should be at least 1 for
       migratable".
 ::= {mcEntry 11}
```

mcMgmtOp OBJECT-TYPE

```
   SYNTAX INTEGER {
        START(0),
        SUSPEND (1),
        RESUME (2),
          STOP(3),
          DESTROY(4)
       }
     ACCESS read-write
     STATUS mandatory
     DESCRIPTION
         "Set to one of the possible management operation a mobile code can
         accept if  implements the  Callback interface".
 ::= {mcEntry 12}


-- mhMIB group
-- This MIB group defines the migration history of these mobile codes running in the
-- system.
-- The system  scope could be a local mobile code system (a Mobile Code Daemon )or
-- a  mobile code region (a Mediator ).


mcMigrationHistoryTable OBJECT-TYPE
     SYNTAX SEQUENCE  OF MCMigrationHistoryEntry
     ACCESS not-accesible
     STATUS mandatory
     DESCRIPTION
         "A list of the mobile codes visited nodes."
   ::= {mhMIB 1}
```

mcMigrationHistoryEntry OBJECT-TYPE

SYNTAX MCMigrationHistoryEntry

ACCESS not-accesible

STATUS mandatory

DESCRIPTION

"Objects related to the attributes of a mobile code's visited node.

An entry in the table is uniquely identified by two attributes: the mobile code

index, corresponding the index in the mcTable and the migration history index"

INDEX { mcIndex, mhIndex}

::= {mcMigrationHistoryTable 1}

MCMigrationHistoryEntry ::= SEQUENCE {

mcIndex

INTEGER,

mhIndex

INTEGER,

mhLocation

DisplayString(SIZE(0..80))

}

mcIndex OBJECT-TYPE

SYNTAX INTEGER

```
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "A unique value identifying the mobile code in the mcTable.
         It ranges between 0 and the value of mcNumber."
   ::= {mcMigrationHistoryEntry 1}

mhIndex  OBJECT-TYPE
    SYNTAX INTEGER
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
         "A unique number indentifying the visited node in a mobile code migration
          history.
          mhIndex is part of  the mcMigrationHistoryEntry index"
   ::= { mcMigrationHistoryEntry 2 }

mhNode OBJECT-TYPE
    SYNTAX DisplayString(SIZE(0..80))
    ACCESS read-only
    STATUS mandatory
    DESCRIPTION
        "mhNode is the MobileCodeDaemon location this mobile code has visited or are
```

visiting.

"The format uniquely identifies a visited node in a region.

hostaddress:tcpPort:udpPort:facPort

hostaddress could be the host IP address or its full domain name.

tcpPort is the TCP port the daemon is listening for incoming requests.

udpPort is the UDP port the daemon is listening for incoming requests.

facPort is the TCP port the daemon's communication facilitator(CF) uses."

::= { mcMigrationHistoryEntry 3 }

Appendix C

Management Request

The figure bellow displays an example of the interaction among components when an SNMP manager issues a set request in a regional management scenario:

1. The SNMP manager issues an SNMP set request to *suspend* the MC21 mobile code running on the remote *MCD02*.

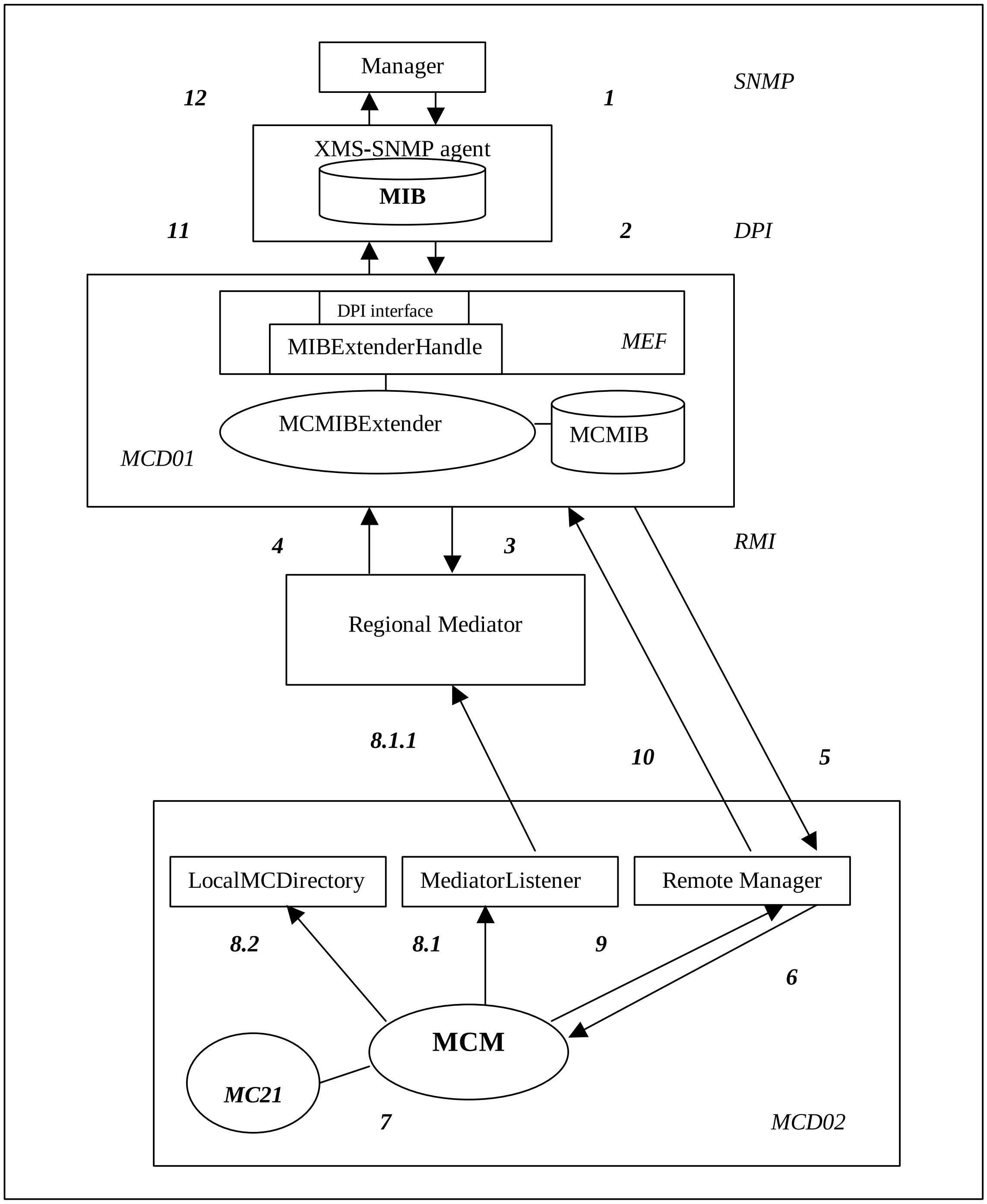
2. Then, the *XMS-SNMP agent* forward the request, using its DPI interface, to the *MIBExtenderHandler* through wich the *MCMIBExtender* is regististered.

3. The *MCMIBExtender* issues an RMI request to the *Mediator* to get the up-to-date version of the regional mobile code database.

4. The *MCMIBExtender* receives the copy and then, it locates where, in the region, the specified mobile code is currently running and whether this remote daemon accepts remote management operation on its mobile codes.

5. If that's the case, it sends the management operation to the *Remote Manager*.

6. As soon as the Remote Manager receives the request, it forwards such request to the local MCM.


Manager
12
1
SNMP
XMS-SNMP agent
MIB
11
2
DPI
DPI interface
MIBExtenderHandle
MEF
MCMIBExtender
MCMIB
MCD01
4
3
RMI
Regional Mediator
8.1.1
10
5
LocalMCDirectory
MediatorListener
Remote Manager
8.2
8.1
9
6
MCM
MC21
7
MCD02

7. Then, the MCM calls the corresponding method on the corresponding mobile code.

8. A change in the execution mobile code status is broadcasted by the MCM's event dispatcher to all the interested parts.

9. And a response is sent back to the local *RemoteManager*.

10. The *MCMIBExtender* receives the results from the RemoteManager and,

11. It sends the rsponse back to the *XMS-SNMP agent* via the DPI interface.

12. Eventually, the manager gets the results and he(she) could check up on the execution status of the corresponding mobile code.

Appendix D

Local Mobile Agent MIB Test

This appendix also serves as a guide for the use of the MCT- *XMS-SNMP agent* integrated environment. It covers the MCE configuration and startup with a local mobile agent MIB.

The details of the *XMS-SNMP agent* configuration is not covered in this appendix instead, it can be found in [55].

Starting up the Mediator and the ND02 netlet on inm-057178

On inm-057178, the MCT is installed on the E:\mctoolkit directory.

1. Running the NT *rmiregistry*. Click on the *ntreg.bat* file from the NT Explorer.

```
ECHO OFF
REM Create a MCToolkit Environment on NT
REM  Phase 1: Run nt rmiregistry
ECHO ON

set jdk=E:\JBuilder3\java
set PATH=%jdk%\bin;

REM  PC20000513 - Run the NT rmiregistry process
set CLASSPATH=
%jdk%\bin\rmiregistry
```

2. Running the Mediator in a separated JVM. Click on the *ntmediator.bat* file from the NT Explorer.

```
ECHO OFF
REM Create a MCToolkit Environment on NT
REM Run the Regional Database on  NT
ECHO ON

set jdk=E:\JBuilder3\java
set PATH=%jdk%\bin;
set mctoolkit=E:\mctoolkit


REM  PC20000513 - Run regional Database: RemoteMediator
java  mct.mediator.RemoteMediator
%mctoolkit%\mct\configuration\dir.properties
```

3. Running the *ND02* netlet daemon. Click on the *ntnd02.bat* file from the NT Explorer.

```
ECHO OFF
REM Create a MCToolkit Environment on NT
REM  Phase 2: Run the NT  RMINetletDaemon
ECHO ON

set jdk=E:\JBuilder3\java
set PATH=%jdk%\bin;
set mctoolkit=E:\mctoolkit

REM  PC20000513 -  Run  Mobile Code Daemon :  nd02
java   mct.RMINetletDaemon
%mctoolkit%\mct\configuration\ntnd02.properties
```

Starting up the XMS-SNMP agent and ND01 netlet on sunspot.

1. Running the *XMS-SNMP agent* daemon. Click on *snmpdsh* tcsh script from the *File Manager* window.

```
#/bin/tcsh

#PC19991220 – Run the xms-snmp agent daemon

~patricia/version-sept98/xms-snmp-sept98/agent /snmpd -f -d -a -q
-p 1061 -l ~patricia/test/snmpd/snmpd.log &
```

2. Running the *ND01* netlet daemon and the client application. Click on the *lsunspot* tcsh script. It launches the *rmiregistry* to run in the background, the *ND01* netlet daemon, and finally the client application to dynamically inject mobile agents into the MCE.

```
#!/bin/tcsh

set jdk=/usr/local/jdk1.2
set mctoolkit=/home/patricia/mctoolkit
setenv  PATH $jdk/bin:$PATH
setenv CLASSPATH .:/$jdk/lib/tools.jar:$mctoolkit/

# PC19991202 - Run the sparc rmiregistry process
rmiregistry &

# PC19991202 - Create a Mobile Code Region : Phase 2
# this c-shell script runs  Mobile Code Daemons : nd01
java   mct.RMINetletDaemon
$mctoolkit/mct/configuration/lsunspot.properties &

# PC19991202 - Runs the client application to inject mobile code:
WinMCTRMIClient
java  mct.clients.WinMCTRMIClient
$mctoolkit/mct/configuration/client01.properties
```

The *lsunspot.properties* file states the default next hop in the migration path, configures the MIB extension facility for mobile agents wishing to act as XMS-SNMP subagents, the *MIBExtendFacilitator* identified as *MEF01*. Finally, configure the *MCMIBExtender* identified as *MCMIB01* and the Local Manager, *LMGR01*.

```
netletdaemon.id=ND01
netletdaemon.console=true
netletdaemon.console.display.errors=true
netletdaemon.console.display.warnings=true
netletdaemon.console.display.messages=true
netletdaemon.console.display.debug=false
netletdaemon.console.display.information=true
netletdaemon.console.display.application=false
netletdaemon.default.protocol=rmi
netletdaemon.facilitator.enable=true
netletdaemon.facilitator.mobilecode=CF01@mct.mediator.CommunicationFacilitator
netletdaemon.facilitator.properties=/home/patricia/mctoolkit/mct/configuration/fac01.properties
netletdaemon.migrator.enable=true
netletdaemon.migrator.mobilecode=MF01@mct.admin.RemoteMigrationFacilitator
netletdaemon.migration.rmi.host.0=inm-057178.sce.carleton.ca
netletdaemon.migration.rmi.name.0=MF02@mct.admin.RemoteMigrationFacilitator
netletdaemon.security.enable=false
########################################################################
#           MIB Extend Facilitator Configuration
########################################################################
netletdaemon.mibextend.enable=true
netletdaemon.mibextend.vmc=MEF01@mct.users.MIBExtendFacilitator

########################################################################
#           Mobile Code  MIB Extender Configuration
########################################################################
netletdaemon.install.mobilecode.0=MCMIB01@mct.mcmgmt.MCMIBExtender
netletdaemon.install.mobilecode.properties.0=/home/patricia/mctoolkit/mct/configuration
/lmcmib.properties
netletdaemon.install.mobilecode.1=LMGR01@mct.management.LocalManager
```

MCMIB01 owns a one-line properties file, *lmcmib.properties*, which states the scope of the mobile agent management:

```
mcmibextender.management=local
```

Starting up ND03 netlet on inm-057179

1. Running the *rmiregistry* . Click on the *ntreg.bat* file from the NT Explorer.

```
ECHO OFF
REM Create a MCToolkit Environment on NT
REM  Phase 1: Run nt rmiregistry
ECHO ON

set jdk=C:\JBuilder3\java
set PATH=%jdk%\bin;

REM  PC20000513 - Run the NT rmiregistry process
set CLASSPATH=
%jdk%\bin\rmiregistry
```

2. Running the *ND03* netlet daemon. Click on the *ntnd03.bat* batch file from the NT Explorer.

```
ECHO OFF
REM Create a MCToolkit Environment on NT
REM  Phase 2: Run the NT  RMINetletDaemon
ECHO ON

set jdk=C:\JBuilder3\java
set PATH=%jdk%\bin;
set mctoolkit=E:\mctoolkit

REM  PC20000513 -  Run  Mobile Code Daemon :  nd03
java   mct.RMINetletDaemon
%mctoolkit%\mct\configuration\ntnd02.properties
```

Querying the MIB

The *snmptest* manager application and the objects and object instance identifiers illustrated in Table D.1 and Table D.2 are used to interrogate the *XMS-SNMP agent*'s MIB.

According to its position in the OSI tree-structured MIB, the MCMIBTREE group identifier is 1.3.6.1.3.100.3, therefore all the object identifiers in this group are of the form

Object	**Object IDentifier (*OID*)**
MCMIB	x.1
mcNumber	x.1.1
mcTable	x.1.2
mcEntry	x.1.2.1
mcIndex	x.1.2.1.1
mcId	x.1.2.1.2
mcClassName	x.1.2.1.3
mcAlias	x.1.2.1.4
mcType	x.1.2.1.5
mcInfo	x.1.2.1.6
mcLocation	x.1.2.1.7
mcStatus	x.1.2.1.8
mcMessagingAccess	x.1.2.1.9
mcMigratable	x.1.2.1.10
mcVisitedNodes	x.1.2.1.11
mcMgmtOp	x.1.2.1.12
MHMIB	x.2
mcMigrationHistoryTable	x.2.1
mcMigrationHistoryEntry	x.2.1.1
mcIndex	x.2.1.1.1
mhIndex	x.2.1.1.2
mhNode	x.2.1.1.3

Table D.1 Object Identifiers for MCMIBTREE Group

where x = 1.3.6.1.3.100.3, the MCMIBTREE group identifier.

And the instance identifiers for MCMIBTREE table entries are

Table	**Row identifier**	**OID**
mcTable	1.3.6.1.3.100.3.1.2.1	*y*.*i*.(mcIndex)
mcMigrationHistoryTable	1.3.6.1.3.100.3.2.1.1	*y*.*j*.(mcIndex).(mhIndex)

Table D.2 Instance Identifiers for the MCMIBTREE Table Entries

where

y = row identifier;

i = columnar object identifier in the range 1..12;

j = columnar identifier in the range 1..3 and

the format (mcIndex) and (mhIndex) indicates the values of these objects respectively.

The first GetRequest is issued to retrieve the current number of mobile agents residing in *ND01*,

GetRequest (mcNumber.0)

then, a response is returned and displayed with value =6

GetResponse(mcNumber.0=6)

The second request wishes to retrieve the mobile code descriptive identifiers of these six mobile agents visiting *ND01*, GetRequest (mcId.1, mcId.2, mcId.3, mcId.4, mcId.5, mcId.6)

and the response received is

```
GetResponse (
mcId.1="MEF01[PUBLIC]@mct.users.MIBExtendFacilitator",
mcId.2="MCMIB01[PUBLIC]@mct.mcmgmt.MCMIBExtender",
mcId.3="MF01[PUBLIC]@mct.admin.RemoteMigrationFacilitator",
mcId.4="RML01[PUBLIC]@mct.mediator.RemoteMediatorListener",
mcId.5="MyHello01[PUBLIC]@mct.examples.Hello.Hello",
mcId.6="LMGR01[PUBLIC]@mct.management.LocalManager")
```

The final request retrieves a complete row in the mcTable

```
GetRequest (mcIndex.5, mcId.5, mcClassName.5, mcAlias.5, mcType.5,
        mcInfo.5, mcLocation.5, mcStatus.5, mcMessagingAccess.5,
        mcMigratable.5, mcVisistedNodes.5,mcMgmtOp.5)
```

and the response received is

```
GetResponse (mcIndex.5=5,
        mcId ="MyHello01[PUBLIC]@mct.examples.Hello.Hello",
        mcClassName= "mct.examples.Hello.Hello",
mcAlias= "mct.examples.Hello.Hello"
mcType= "NETLET",
mcInfo= "No information available",
mcLocation= "134.117.4.14:-1:-1:6668",
mcStatus= "RUNNING",
mcMessagingAccess= "No "
mcMigratable "Yes"
mcVisitedNodes= 1
mcMgmtOp= -1)
```

The Figure D.1 displays the manager application interaction with the *XMS-SNMP agent*.

```
Variable: 1.3.6.1.3.100.3.1.1.0
Variable:
Received Get Response from 134.117.4.14
requestid 0x39477187 errstat 0x0 errindex 0x0
100.3.1.0.1.0 6
Variable: 1.3.6.1.3.100.3.1.2.1.2.1
Variable: 1.3.6.1.3.100.3.1.2.1.2.2
Variable: 1.3.6.1.3.100.3.1.2.1.2.3
Variable: 1.3.6.1.3.100.3.1.2.1.2.4
Variable: 1.3.6.1.3.100.3.1.2.1.2.5
Variable: 1.3.6.1.3.100.3.1.2.1.2.6
Variable:
Received Get Response from 134.117.4.14
requestid 0x39477188 errstat 0x0 errindex 0x0
100.3.1.2.1.2.1 "MEF01[PUBLIC]@mct.users.MIBExtendFacilitator"
100.3.1.2.1.2.2 "MCMIB01[PUBLIC]@mct.mcmgmt.MCMIBExtender"
100.3.1.2.1.2.3 "MF01[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.4 "RML01[PUBLIC]@mct.mediator.RemoteMediatorListener"
100.3.1.2.1.2.5 "MyHello01[PUBLIC]@mct.examples.Hello.Hello"
100.3.1.2.1.2.6 "LMGR01[PUBLIC]@mct.management.LocalManager"
Variable: 1.3.6.1.3.100.3.1.2.1.1.5
Variable: 1.3.6.1.3.100.3.1.2.1.2.5
Variable: 1.3.6.1.3.100.3.1.2.1.3.5
Variable: 1.3.6.1.3.100.3.1.2.1.4.5
Variable: 1.3.6.1.3.100.3.1.2.1.5.5
Variable: 1.3.6.1.3.100.3.1.2.1.6.5
Variable: 1.3.6.1.3.100.3.1.2.1.7.5
Variable: 1.3.6.1.3.100.3.1.2.1.8.5
Variable: 1.3.6.1.3.100.3.1.2.1.9.5
Variable: 1.3.6.1.3.100.3.1.2.1.10.5
Variable: 1.3.6.1.3.100.3.1.2.1.11.5
Variable: 1.3.6.1.3.100.3.1.2.1.12.5
Variable:
Received Get Response from 134.117.4.14
requestid 0x39477189 errstat 0x0 errindex 0x0
100.3.1.2.1.1.5 5
100.3.1.2.1.2.5 "MyHello01[PUBLIC]@mct.examples.Hello.Hello"
100.3.1.2.1.3.5 "mct.examples.Hello.Hello"
100.3.1.2.1.4.5 "mct.examples.Hello.Hello"
100.3.1.2.1.5.5 "NETLET"
100.3.1.2.1.6.5 "No information available"
100.3.1.2.1.7.5 "134.117.4.14:-1:-1:6668"
100.3.1.2.1.8.5 "RUNNING"
100.3.1.2.1.9.5 "No "
100.3.1.2.1.10.5 "Yes"
100.3.1.2.1.11.5 1
100.3.1.2.1.12.5 -1
Variable: $Q
Quitting,   Goodbye
```

Figure D.1 snmptest Application Interaction with the XMS-SNMP agent

Appendix E

Regional Mobile Agent MIB Test

In a regional mode management scenario, the network manager application queries the *XMS-SNMP agent* and will be able to manage all mobile agents running in the region. In this case, the *ND01* is configured with a *MIBExtendFacilitator* and a *MCMIBExtender*, which extends the *XMS-SNMP agent*'s MIB, interfacing the mobile agents visiting the three MCDs currently registered with the Mediator at *sunspot*, *inm-057178* and *inm-057179* respectively. It also requires the three netlet daemons configure the *RemoteManager* component to accept management operations coming from the remote network manager station.

For *ND01*, the following lines are included in the netlet daemon's properties file,

```
#################################################################
#              MIB Extend Facilitator Configuration
#################################################################
netletdaemon.mibextend.enable=true
netletdaemon.mibextend.vmc=MEF01@mct.users.MIBExtendFacilitator
##################################################################
#              Mobile Code  MIB Extender Configuration
##################################################################
```

```
netletdaemon.install.mobilecode.0=MIB01@mct.mcmgmt.MCMIBExtender
netletdaemon.install.mobilecode.properties.0=/home/patricia/mctoolkit/mct/co
nfiguration/mcmib.properties
netletdaemon.install.mobilecode.1=RM01@mct.management.RemoteManager
```

where **mcmib.properties** is a one-line test file containing

```
mcmibextender.management=regional
```

For *ND02,*

```
netletdaemon.install.mobilecode.1=RM02@mct.management.RemoteManager
```

and for *ND03*

```
netletdaemon.install.mobilecode.1=RM03@mct.management.RemoteManager
```

Following the same configuration sequence than the local management scenario above, the *Mediator* and *ND02* netlet daemon will be started up in *inm-057178* first, followed by the *XMS-SNMP agent* and the *ND01* netlet daemon in *sunspot* and *ND03* netlet daemon in *inm-057179* will start third.

The *snmptest* manager application is used to interrogate the *XMS-SNMP agent*'s MIB.

The first request retrieves the number of mobile agents currently registered in the region:

> GetRequest (mcNumber.0)

then, a response is returned and displayed with value =10

> GetResponse(mcNumber.0=10)

The second request wishes to retrieve the mobile code descriptive identifiers of these 10 mobile agents,

> GetRequest (mcId.1, mcId.2, mcId.3, mcId.4, mcId.5, mcId.6,
> mcId.7, mcId.8, mcId.9, mcId.10)

and the response received is

```
GetResponse (
mcId.1=
"MyTicTacToe01[PUBLIC]@mct.examples.McTicTacToe.McTicTacToe",
     mcId.2= "RMGR02[PUBLIC]@mct.management.RemoteManager",
     mcId.3= "RMGR01[PUBLIC]@mct.management.RemoteManager",
     mcId.4= "RMGR03[PUBLIC]@mct.management.RemoteManager",
     mcId.5= "MF03[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
     mcId.6= "MF02[PUBLIC]@mct.admin.RemoteMigrationFacilitator",
     mcId.7=
     "MyHello02[PUBLIC]@mct.examples.Hello.Hello",
     mcId.8= "MF01[PUBLIC]@mct.admin.RemoteMigrationFacilitator",
     mcId.9=
     "MCMIB01[PUBLIC]@mct.mcmgmt.MCMIBExtender",
     mcId.10=
     "MEF01[PUBLIC]@mct.users.MIBExtendFacilitator" )
```

The third request retrieves a complete row in the mcTable

```
     GetRequest (mcIndex.7, mcId.7, mcClassName.7, mcAlias.7, mcType.7,
          mcInfo.7,   mcLocation.7,   mcStatus.7,   mcMessagingAccess.7,
          mcMigratable.7, mcVisistedNodes.7,mcMgmtOp.7)
```

and receives:

```
  GetResponse (mcIndex.7=7,
            mcId
         ="MyTicTacToe01[PUBLIC]@mct.examples.McTicTacToe.McTicTacToe",
            mcClassName= " mct.examples.McTicTacToe.McTicTacToe",
            mcAlias= " mct.examples.McTicTacToe.McTicTacToe",
            mcType= "NETLET",
            mcInfo= "No information available",
            mcLocation= "134.117.57.178:-1:-1:7777",
            mcStatus= "RUNNING",
            mcMessagingAccess= "No "
            mcMigratable "Yes"
            mcVisitedNodes= 8
            mcMgmtOp= -1)
```

To retrieve the migration history of the mobile agent above, the get request looks like:

```
            GetRequest (mhNode.7.1.1, mhNode.7.1.2., mhNode.7.1.3, mhNode.7.1.4,
                    mhNode.7.1.5, mhNode.7.1.6, mhNode.7.1.7, mhNode.7.1.8 )
```

and it receives,

```
GetResponse(mhNode.7.1.1= "//sunspot.sce.carleton.ca:-1:-:6666/ND01",
        mhNode.7.1.2= "//inm-057178:-1:-1:7777/ND02",
        mhNode.7.1.3= "//inm-057179:-1:-1:7778/ND03" ,
        mhNode.7.1.4= "//sunspot.sce.carleton.ca:-1:-1:6666/ND01",
        mhNode.7.1.5= "//inm-057178:-1:-1:7777/ND02",
        mhNode.7.1.6= "//inm-057179:-1:-1:7778/ND03",
        mhNode.7.1.7= "//sunspot.sce.carleton.ca:-1:-1:6666/ND01" ,
        mhNode.7.1.8= "//inm-057178:-1:-1:7777/ND02" )
```

To send a management operation to stop an mobile agent in the region, the network manager will

```
SetRequest (mcMgmtOp.7, i, 3)
```

and receives

```
GetResponse(mcMgmtOp.7=3)
```

Then, the mobile agent's operational status can be retrieved issuing

```
GetRequest(mcStatus.7)
```

and the response is

GetResponse(mcStatus.7=”STOPPED”)

But this mobile agent is still in *ND02* and registered with the Mediator. To completely destroy it, the network manager can:

SetRequest (mcMgmtOp.7, i, 4)

obtaining

GetResponse(mcMgmtOp.7=4)

and the number of mobile agents now registered with the Mediator

GetRequest(mcNumber.0)

will be updated to

GetResponse(mcNumber.0=9)

The complete *snmptest* interaction with the *XMS-SNMP agent* can be shown in Figure E.1.

```
Variable: 1.3.6.1.3.100.3.1.1.0
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE15B errstat 0x0 errindex 0x0
100.3.1.0.1.0 10
Variable: 1.3.6.1.3.100.3.1.2.1.2.1
Variable: 1.3.6.1.3.100.3.1.2.1.2.2
Variable: 1.3.6.1.3.100.3.1.2.1.2.3
Variable: 1.3.6.1.3.100.3.1.2.1.2.4
Variable: 1.3.6.1.3.100.3.1.2.1.2.5
Variable: 1.3.6.1.3.100.3.1.2.1.2.6
Variable: 1.3.6.1.3.100.3.1.2.1.2.7
Variable: 1.3.6.1.3.100.3.1.2.1.2.8
Variable: 1.3.6.1.3.100.3.1.2.1.2.9
Variable: 1.3.6.1.3.100.3.1.2.1.2.10
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE15C errstat 0x0 errindex 0x0
100.3.1.2.1.2.1 "MyTicTacToe01[PUBLIC]@mct.examples.McTicTacToe.McTicTacToe"
100.3.1.2.1.2.2 "RMGR02[PUBLIC]@mct.management.RemoteManager"
100.3.1.2.1.2.3 "RMGR01[PUBLIC]@mct.management.RemoteManager"
100.3.1.2.1.2.4 "RMGR03[PUBLIC]@mct.management.RemoteManager"
100.3.1.2.1.2.5 "MF03[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.6 "MF02[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.7 "MyHello02[PUBLIC]@mct.examples.Hello.Hello"
100.3.1.2.1.2.8 "MF01[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.9 "MCMIB01[PUBLIC]@mct.mcmgmt.MCMIBExtender"
100.3.1.2.1.2.10 "MEF01[PUBLIC]@mct.users.MIBExtendFacilitator"
Variable: 1.3.6.1.3.100.3.1.2.1.1.1
Variable: 1.3.6.1.3.100.3.1.2.1.2.1
Variable: 1.3.6.1.3.100.3.1.2.1.3.1
Variable: 1.3.6.1.3.100.3.1.2.1.4.1
Variable: 1.3.6.1.3.100.3.1.2.1.5.1
Variable: 1.3.6.1.3.100.3.1.2.1.6.1
Variable: 1.3.6.1.3.100.3.1.2.1.7.1
Variable: 1.3.6.1.3.100.3.1.2.1.8.1
Variable: 1.3.6.1.3.100.3.1.2.1.9.1
Variable: 1.3.6.1.3.100.3.1.2.1.10.1
Variable: 1.3.6.1.3.100.3.1.2.1.11.1
Variable: 1.3.6.1.3.100.3.1.2.1.12.1
Variable:
Received Get Response from 134.117.4.14
100.3.1.2.1.1.1 1
100.3.1.2.1.2.1 "MyTicTacToe01[PUBLIC]@mct.examples.McTicTacToe.McTicTacToe"
100.3.1.2.1.3.1 "mct.examples.McTicTacToe.McTicTacToe"
100.3.1.2.1.4.1 "mct.examples.McTicTacToe.McTicTacToe"
100.3.1.2.1.5.1 "NETLET"
100.3.1.2.1.6.1 "No information available"
100.3.1.2.1.7.1 "134.117.57.178:-1:-1:7777"
100.3.1.2.1.8.1 "RUNNING"
100.3.1.2.1.9.1 "No "
100.3.1.2.1.10.1 "Yes"
100.3.1.2.1.11.1 8
100.3.1.2.1.12.1 -1
```

```
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.1
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.2
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.3
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.4
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.5
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.6
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.7
Variable: 1.3.6.1.3.100.3.2.1.1.3.1.8
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE15E errstat 0x0 errindex 0x0
100.3.2.1.1.3.1.1 "//sunspot.sce.carleton.ca:-1:-1:6666/ND01"
100.3.2.1.1.3.1.2 "//inm-057178:-1:-1:7777/ND02"
100.3.2.1.1.3.1.3 "//inm-057179:-1:-1:7778/ND03"
100.3.2.1.1.3.1.4 "//sunspot.sce.carleton.ca:-1:-1:6666/ND01"
100.3.2.1.1.3.1.5 "//inm-057178:-1:-1:7777/ND02"
100.3.2.1.1.3.1.6 "//inm-057179:-1:-1:7778/ND03"
100.3.2.1.1.3.1.7 "//sunspot.sce.carleton.ca:-1:-1:6666/ND01"
100.3.2.1.1.3.1.8 "//inm-057178:-1:-1:7777/ND02"
Variable: 1.3.6.1.3.100.3.1.2.1.1.7
Variable: 1.3.6.1.3.100.3.1.2.1.2.7
Variable: 1.3.6.1.3.100.3.1.2.1.3.7
Variable: 1.3.6.1.3.100.3.1.2.1.4.7
Variable: 1.3.6.1.3.100.3.1.2.1.5.7
Variable: 1.3.6.1.3.100.3.1.2.1.6.7
Variable: 1.3.6.1.3.100.3.1.2.1.7.7
Variable: 1.3.6.1.3.100.3.1.2.1.8.7
Variable: 1.3.6.1.3.100.3.1.2.1.9.7
Variable: 1.3.6.1.3.100.3.1.2.1.10.7
Variable: 1.3.6.1.3.100.3.1.2.1.11.7
Variable: 1.3.6.1.3.100.3.1.2.1.12.7
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE15F errstat 0x0 errindex 0x0
100.3.1.2.1.1.7 7
100.3.1.2.1.2.7 "MyHello02[PUBLIC]@mct.examples.Hello.Hello"
100.3.1.2.1.3.7 "mct.examples.Hello.Hello"
100.3.1.2.1.4.7 "mct.examples.Hello.Hello"
100.3.1.2.1.5.7 "NETLET"
100.3.1.2.1.6.7 "No information available"
100.3.1.2.1.7.7 "134.117.57.178:-1:-1:7777"
100.3.1.2.1.8.7 "RUNNING"
100.3.1.2.1.9.7 "No "
100.3.1.2.1.10.7 "Yes"
100.3.1.2.1.11.7 1
100.3.1.2.1.12.7 -1
```

```
Variable: $S
Request type is Set Request
Variable: 1.3.6.1.3.100.3.1.2.1.12.7
Type [i|s|x|d|n|o|t|a]: i
Value: 3
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE160 errstat 0x0 errindex 0x0
100.3.1.2.1.12.7 3
Variable: $G
Request type is Get Request
Variable: 1.3.6.1.3.100.3.1.2.1.8.7
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE161 errstat 0x0 errindex 0x0
100.3.1.2.1.8.7 "STOPPED"
Variable: $S
Request type is Set Request
Variable: 1.3.6.1.3.100.3.1.2.1.12.7
Type [i|s|x|d|n|o|t|a]: i
Value: 4
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE162 errstat 0x0 errindex 0x0
100.3.1.2.1.12.7 4
Variable: $G
Request type is Get Request
Variable: 1.3.6.1.3.100.3.1.1.0
Variable:
Received Get Response from 134.117.4.14
requestid 0x1CBAE163 errstat 0x0 errindex 0x0
100.3.1.0.1.0 9
Variable: 1.3.6.1.3.100.3.1.2.1.2.1
Variable: 1.3.6.1.3.100.3.1.2.1.2.1
Variable: 1.3.6.1.3.100.3.1.2.1.2.2
Variable: 1.3.6.1.3.100.3.1.2.1.2.3
Variable: 1.3.6.1.3.100.3.1.2.1.2.4
Variable: 1.3.6.1.3.100.3.1.2.1.2.5
Variable: 1.3.6.1.3.100.3.1.2.1.2.6
Variable: 1.3.6.1.3.100.3.1.2.1.2.7
Variable: 1.3.6.1.3.100.3.1.2.1.2.8
Variable: 1.3.6.1.3.100.3.1.2.1.2.9
Variable:
```

```
Received Get Response from 134.117.4.14
requestid 0x1CBAE164 errstat 0x0 errindex 0x0
100.3.1.2.1.2.1 "MF03[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.2 "MyTicTacToe01[PUBLIC]@mct.examples.McTicTacToe.McTicTacToe"
100.3.1.2.1.2.3 "MF01[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.4 "MF02[PUBLIC]@mct.admin.RemoteMigrationFacilitator"
100.3.1.2.1.2.5 "RMGR02[PUBLIC]@mct.management.RemoteManager"
100.3.1.2.1.2.6 "MEF01[PUBLIC]@mct.users.MIBExtendFacilitator"
100.3.1.2.1.2.7 "MCMIB01[PUBLIC]@mct.mcmgmt.MCMIBExtender"
100.3.1.2.1.2.8 "RMGR01[PUBLIC]@mct.management.RemoteManager"
100.3.1.2.1.2.9 "RMGR03[PUBLIC]@mct.management.RemoteManager"
Variable: $Q
Quitting,   Goodbye
```

Figure E.1 snmptest Application Interrogating the XMS-SNMP agent's MIB