Contents

Contents........................................................................................................................................ b
Figures ........................................................................................................................................ d
Acronyms ...................................................................................................................................... f
1. The SNMP protocol and SnmpWorkShop ................................................................................. 1
  1.1 Introduction ............................................................................................................................ 1
  1.2 The SNMP protocol ................................................................................................................ 2
    1.2.1 The management model of SNMP .................................................................................. 2
    1.2.2 The fonctions of the protocol .......................................................................................... 3
    1.2.3 Remark on management information roles ................................................................. 3
  1.3 The SNMPWorkShop toolbox ............................................................................................... 4
    1.3.1 General overview ............................................................................................................ 4
    1.3.2 SNMPWS development .................................................................................................. 5
2. Practice ...................................................................................................................................... 7
  2.1 Development environnement ................................................................................................. 7
    2.1.1 Prerequisites .................................................................................................................. 7
    2.1.2 The development cycle .................................................................................................. 7
  2.2 Architecture overview ........................................................................................................... 8
  2.3 The agent ................................................................................................................................. 9
  2.4 The manager .......................................................................................................................... 11
  2.5 Additional features .............................................................................................................. 11
Annex: Initial coffee pot specifications ....................................................................................... 12
Notes ........................................................................................................................................... 15
Figures

Figure 1: simplified overview of management.......................................................... 1
Figure 2: the SNMP frame .................................................................................. 2
Figure 3: MIB structure ...................................................................................... 3
Figure 4: management information roles ............................................................ 4
Figure 5: SNMPWS architecture ....................................................................... 4
Figure 6: the two phases of SNMPWS development ........................................ 5
Figure 7: Lab development cycle ...................................................................... 7
Figure 8: management architecture of the coffee pot system............................ 8
Figure 9: link with generic model ...................................................................... 8
Figure 10: scheme of the agent ......................................................................... 10
Figure 11: algorithm of the manager ................................................................. 11
Acronyms

API  Application Programming Interface
ASN.1  Abstract Syntax Notation One
ATM  Asynchronous Transfer Mode
B2B  Business to Business
B-ISDN  Broadband ISDN
BOOTP  Boot protocol
CLI  Command Line Interface
CMIP  Common Management Information Protocol
CMIP/GDMO  Common Management Information Protocol/Guidelines for the Definition of Managed Objects
COPS  Common Open Policy Service
COPS-PR  COPS Usage for Policy Provisioning
CORBA IIOP  Common Object Request Broker Architecture Internet Inter-ORB Protocol
CORBA  Common Object Request Broker Architecture
CORBA/IDL  Common Object Request Broker Architecture/Interface Definition Language
DCN  Data Communications Network
DECT  Digital Enhanced Cordless Telecommunications
DHCP  Dynamic Host Configuration Protocol
DNS  Directory Name Service
DSS1  Digital Subscriber System 1
EM  Element Manager
EMS  Element Management System
FFS  For Further Study
FTAM  File Transfer Access and Management
FTP  File Transfer Protocol
ftp  FTP
GDMO  Guidelines for the Definition of Managed Objects
GGSN  Gateway GPRS Support Node
Go interface  The interface between the GGSN and the Policy Decision Function (PDF)
GSM  Global System for Mobile communications
HLR  Home Location Register
HSS  Home Subscriber Server
IDL  Interface Definition Language
IETF  Internet Engineering Task Force
IIOP  Internet Inter-ORB Protocol
IN  Intelligent Network
INAP  Intelligent Network Application Part
IRP  Integration Reference Point
IS  Information Service
ISDN  Integrated Services Digital Network
JDMK  Java Dynamic Management toolKit
LDAP  Lightweight Directory Access Protocol
LDUP  LDAP Duplication/Replication/Update Protocols
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLA</td>
<td>Logical Layered Architecture</td>
</tr>
<tr>
<td>MAP</td>
<td>Mobile Application Part</td>
</tr>
<tr>
<td>MExE</td>
<td>Mobile Execution Environment</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>MMI</td>
<td>Man-Machine Interface</td>
</tr>
<tr>
<td>NM</td>
<td>Network Manager</td>
</tr>
<tr>
<td>NMS</td>
<td>Network Management System</td>
</tr>
<tr>
<td>NRM</td>
<td>Network Resource Model</td>
</tr>
<tr>
<td>OS</td>
<td>Operations System</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>OSS</td>
<td>Operations Support System</td>
</tr>
<tr>
<td>PDF</td>
<td>Policy Decision Function</td>
</tr>
<tr>
<td>PDH</td>
<td>Plesiochronous Digital Hierarchy</td>
</tr>
<tr>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>PIB</td>
<td>Policy Information Base</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>PLMN</td>
<td>Public Land Mobile Network</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RNC</td>
<td>Radio Network Controller</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource ReserVation Protocol</td>
</tr>
<tr>
<td>SDH</td>
<td>Synchronous Digital Hierarchy</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol (IETF)</td>
</tr>
<tr>
<td>SNMP/SMI</td>
<td>SNMP/Structure of Management Information</td>
</tr>
<tr>
<td>SOM</td>
<td>Service Operations Management</td>
</tr>
<tr>
<td>SS</td>
<td>Solution Set</td>
</tr>
<tr>
<td>SS7</td>
<td>Signalling System No. 7</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/ Internet Protocol</td>
</tr>
<tr>
<td>tftp</td>
<td>trivial ftp</td>
</tr>
<tr>
<td>TM</td>
<td>Telecom Management</td>
</tr>
<tr>
<td>TMF</td>
<td>TeleManagement Forum</td>
</tr>
<tr>
<td>TMN</td>
<td>Telecommunications Management Network (ITU-T)</td>
</tr>
<tr>
<td>TOM</td>
<td>Telecom Operations Map (TMF)</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>UPT</td>
<td>Universal Personal Telecommunication</td>
</tr>
<tr>
<td>USIM</td>
<td>Universal Subscriber Identity Module</td>
</tr>
<tr>
<td>UTRA</td>
<td>Universal Terrestrial Radio Access</td>
</tr>
<tr>
<td>VHE</td>
<td>Virtual Home Environment</td>
</tr>
</tbody>
</table>

© INT 2005
1. The SNMP protocol and SnmpWorkShop

1.1 Introduction

Network management facilitates the task to optimize network resources. It is traditionally decomposed into five functional areas FCAPS (cf. courses). In summary, the administrator has to control equipments (routers, concentrators, workstations, etc.) of the network in order to offer the best possible service to users.

The general problem envisioned by the present lab is summarized by figure 1:

![Figure 1: simplified overview of management](image)

In this figure a particular user, the network administrator, manages the equipment from the distance through some management system. This configuration seems quite simple, however it is subject to some unavoidable constraints:

- **Heterogeneousness**: equipments come in very different shapes (capabilities, vendors, protocols, ...); management systems as well;
- **Efficiency**: management has to be relatively efficient (not realtime however) to facilitate end-to-end operations through the whole network;
- **Moreover**, management operations of this layer (EML, cf. courses) should provide means to be automated (cf. NML layer).

In this context vendors pursue a mixed strategy. On one hand some proprietary management solutions are provided; on the other, equipments may implement standard management protocols such as CMIS/P defined by ITU-T or SNMP proposed by IETF. Some implement also CORBA interfaces.

The goal of this lab is to study concretely SNMP management via a particular Java toolbox called SnmpWorkShop.
1.2 The SNMP protocol

1.2.1 The management model of SNMP

SNMP (Simple Network Management Protocol) is the management protocol defined by IETF. It is the de facto universal standard of LAN equipments (however many WAN equipments implement it as well). SNMP is a relatively simple protocol, but its functionalities are powerful enough to provision management of complex heterogeneous networks.

SNMP is also used to manage applications: databases, servers, printers, etc. so its usage may trespass the scope of network management, to concern systems management at large.

The management model of SNMP can be described by the threefold entity called frame (figure 2).

![Figure 2: the SNMP frame](image)

We find there:

- A) The **manager** located in the management system. This entity executes management applications that control network equipments via SNMP requests.
- B) The **agent** is software embedded in the equipment. It is in charge of the control of the equipment resources with respect to management requests issued by the manager via SNMP. The agent maintains a **MIB** (Managed Information Base) i.e. a repository for management information that is a collection of Management Objects.
- C) The managed **resources** i.e. electronics and components implementing the functionalities of the equipment.
1.2.2 The functions of the protocol

1.2.2.1 SNMP primitives

- **get**: allows the manager to interrogate the agent for the value of some MIB object;
- **get_next**: allows to read the next object without knowing its name;
- **set**: allows to modify objects of the MIB;
- **trap**: action to transmit an alarm to the manager.

1.2.2.2 Data model associated to SNMP

- The **SMI** (Structure of Management Information) is a standardized scheme defining access and specification rules. It is the meta model of SNMP.
- The **MIB** gives a common definition of objects and their structure. It is useful to consult standard MIBs such as the MIB-II defined by the RFC1213 document of IETF.

![MIB structure diagram]

*Figure 3: MIB structure*

1.2.3 Remark on management information roles

The MIB plays a twofold role:
• It models the resources that are located inside the agent. So said, the agent implements the MIB (see the frame above);
• It exposes its management objects to the manager that manipulates them via SNMP: the manager possesses knowledge of the MIB.

The initial scheme can therefore be enriched in the following way:

![Figure 4: management information roles](image)

### 1.3 The SNMPWorkShop toolbox

#### 1.3.1 General overview

SnmpWorkShop (SNMPWS) is a toolset allowing efficient development of SNMP-based management solutions. In summary, SNMPWS is a Java library that provides an object-oriented point of view on the SNMP management protocol, as well as tools to facilitate the implementation of agents and managers.

It consists of:

- A framework that handles management processes and provides SNMP stacks; it is implemented by libraries that are linked to the application. The framework has a simple, object-oriented API;
- A compiler called SMIC (SMI Compiler) that plays an essential role: it generates attribute tables for the manager application and the agent, as well as Java skeletons of accessors to the managed objects.

The following figure summarizes the architecture and key concepts of SNMPWS.

![Figure 5: SNMPWS architecture](image)
1.3.2 SNMPWS development

The development process using SNMPWS distinguishes two phases: design and execution.

- During design, the MIB specification is 1) compiled by the SMIC tool to generate Java implementation skeletons of managed objects and attribute tables that will serve both the agent and the manager.
- During execution, the manager sends its requests to the SNMPWS framework that transmits them internally to the concerned MBean; the latter, by its implementation, executes the requests by manipulating equipment resources in a vendor-specific way.

The figure 6 summarizes the two phases of SNMPWS development.

*Figure 6: the two phases of SNMPWS development*
2. Practice

2.1 Development environnement

2.1.1 Prerequisites

It is supposed that the general environment of the workstation has been configured and tested with respect to the instructions available online, according to the available system (Windows or Linux): MIB browser, environment variables PATH and CLASSPATH etc.

Moreover it is supposed that a working directory has been created and loaded with the base files that are: the equipment MIB specifications (CoffeePot.mib), the code of the agent (CoffeePotAgent.java), an implementation squeleton CoffeePotImpl.java, a manager squeleton (CoffeePotManager.java), and an equipment simulator squeleton (CoffeePotSimulator.java).

2.1.2 The development cycle

The development cycle of the lab is summarized by the following figure that shows the different steps and associated tools:

![Figure 7: Lab development cycle](image-url)
2.2 Architecture overview

A reasonable SNMP lab could have been based on an existing standard MIB allowing control of a real network equipment. Nothing stands in the way of this approach that the student may easily carry out after the present exercise. However, giving the priority to pedagogy led to propose a specific MIB designed specially for the lab, without any useless complexity, that models simple equipment (a coffee machine). This choice allows:

- To concentrate on core concepts instead of losing time on the complexity of standard MIBs;
- To master the whole communication chain from the manager to the implementation inside the equipment (which will be simulated by some provided code).

The goal of this lab is therefore to build a realistic coffee machine simulator based on a fully manageable SNMP agent and on a suitable manager. The coffee machine agent will provide following properties:

- Measures for ground coffee, available water, water temperature and others;
- The ability to make coffee on demand through some SNMP requests;
- Choice of coffee options;
- Problem reporting capabilities.

The following figure gives a summary of the coffee pot system. In fact it is an adaptation of the lower part of figure 6.

![Figure 8: management architecture of the coffee pot system](image)

It may be useful to situate this schema with respect to the generic SNMP model described earlier. The figure 9 below makes the links between the two.

![Figure 9: link with generic model](image)
The tasks at hand will now be organized into two main steps: the agent step and the manager step.

### 2.3 The agent

The first step of the lab consists of the development of a working coffee pot SNMP agent. To this purpose a suitable MIB specification is provided (file CoffeePot.mib, see annex 1).

It is useful to read and understand this specification, as this is the first important moment of the lab.

One of the main properties of this specification is that coffee is generated by a set request to the value on, of the attribute CoffeePotControl. Indeed, SNMP does not provide any procedural action call request such as the action() primitive of CMIS/P. This limitation is overrun by our scheme that gives a particular semantics to the on value of this attribute (i.e. make some coffee).

To develop the agent one will proceed by following steps:

- **a)** To test smic, javac, java
- **b)** smic execution and study of the generated code;
- **c)** To build an «empty» agent that is however able to communicate 1) with the manager and then 2) with the implementation;
- **d)** To implement additional or refined functions within the agent.

Let us detail these steps one by one.

**a)** **Test of smic, javac, java**

Check with installation documentation.

**b)** **smic execution**

All Java code skeletons are generated by the command:

```bash
$ smic CoffeePot.mib
```

This command produces two files in the working directory.

It is useful to examine the generated files to get a general idea of how SnmpWorkShop implements its machinery.

Examine the file SMICAccess.java. This file provides all accessors (get and set) of our MIB attributes. This is therefore seemingly a good location to program the intelligence of the agent (in our case, to communicate with the simulator). However please take care:

An additional feature is provided here. Please remind that smic, due to its generative approach, will wipe out any previously generated file i.e. including manually re-engineered files. It is therefore recommended to avoid any manual coding within generated files.

The proposed solution is classical: it consists to create by inheritance a new class that will implement the agent’s intelligence on behalf of the original SMICAccess class; this class will be called SMICAccessImpl. The related file SMICAccessImpl.java will not be in danger of being overwritten. A sample SMICAccessImpl class is provided.

In this context, it has to be noticed that the main program (SMICAgent.java) refers to the class SMICAccess and not our safe class.
SMICAccessImpl. In order to make everything work, this detail has to be taken care for within the code of the main program SMICAgent.java. The line to modify is clearly signalled.

c) 1) communication with the manager
With respect to the development cycle described earlier, it is time to compile all the code with the command:

```bash
$ build_all
```

This should normally result in a working agent, that however lacks any serious functionality. Still this agent can be invoked, with the command:

```bash
$ java CoffeePotAgent
```

To test the SNMP communication, it is obviously necessary to use a manager. The simplest manager at hand is the provided MIB browser. Please launch it and load the MIB specification file CoffeePot.mib. Then connect it to the agent (do not forget to configure the host and the port). Any get and set operation on attributes should now work correctly (however giving some default results); if not, verify previous steps!

The figure below shows a simplified scheme of the agent. Part of it is provided with the lab (file CoffeePotAgent.java), part is generated (the majority). The “intelligence” of the agent (the CoffeePotImpl class) is linked to the CoffeePot class by inheritance.

![Figure 10: scheme of the agent](image)

In this configuration, all processes are located on the same machine. Please verify that everything works also in the distance: launch your agent on another machine and repeat the tests. The MIB browser should connect to it without problems.

c) 2) communication with the simulator
It is now time to implement the communication with the coffee machine simulator CoffeePotSimulator. To do this, write correctly the accessors of the CoffeePotImpl class that shall call the appropriate methods of the simulator. Of course, all tests have to be carried out again, until the complete communication chain is working from the browser to the simulator.

d) Implementation of supplementary functions
When everything is working, there is still the potential to improve the agent with some special functions. As an example, “making” a “cup” of coffee should logically diminish the available water as well as ground coffee. This aspect could be coded within the simulator. As a consequence, the agent should be able to verify the sanity of the coffee machine, and refuse to make coffee if conditions are not appropriate (no more water or ground coffee, too low temperature etc.). A smart agent should also be able to signal such incidents to the manager in an appropriate way (this last aspect i.e. alarm management, is left to further studies).
### 2.4 The manager

In the previous situation, all intelligence of operations was left inside the agent, whereas the manager role was reduced to a simple MIB browser. The interaction with the agent will now be taken in charge by a specific manager that will provide an applicative user interface.

The goal of this part of the lab is therefore to design a manager that works in the following way:

![Figure 11: algorithm of the manager](image)

The manager should be able to verify the sound condition of the coffee machine, before to try to get some coffee. It should also signal any problems, such as low water or ground coffee.

A manager skeleton code is provided.

### 2.5 Additional features

Having obtained good results regarding the manager-agent relationship, new features can now be added to the whole system. Examples that can be mentioned are the choice of making tea, expresso or filter coffee, or the possibility to program the coffee machine with a timer etc. All these features would require enhancements to the specification file of the machine: coffeepot.mib. And as a consequence, the development cycle of figure 7 will start again.

Please remind to save all implementation files with specific file names, since mibgen regenerates all agent and manager files as already mentioned.
Annex: Initial coffee pot specifications

-- This file describes the INT Coffee Pot MIB. Its definition follows here:

COFFEE-POT-MIB DEFINITIONS ::= BEGIN
IMPORTS
    enterprises
FROM RFC1155-SMI OBJECT-TYPE
FROM RFC-1212 DisplayString
FROM RFC-1213;
domestic OBJECT IDENTIFIER ::= {iso(1) org(3) dod(3) internet(1)
    private(4) enterprises(1) 9999 } -- A fictive Private node number
MyCounter ::= INTEGER (0..4294967295)
MyGauge ::= INTEGER (0..4294967295)

-- textual conventions
DisplayString ::= OCTET STRING
    -- This data type is used to model textual information taken
    -- from the NTV ASCII character set. By convention, objects
    -- with this syntax are declared as having
    --
    -- SIZE (0..255)

householdElectricals OBJECT IDENTIFIER ::= {domestic 1}
coffeePot OBJECT IDENTIFIER ::= {householdElectricals 1}

-- coffeePot MIB
-- Ici commence la spécification du coffeePot

coffeePotManufacturer OBJECT-TYPE
SYNTAX DisplayString
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The name of the coffee pot's manufacturer. For instance, ATouteVapeur."
::= {coffeePot 1}

coffeePotModel OBJECT-TYPE
SYNTAX DisplayString
ACCESS read-only
STATUS mandatory
DESCRIPTION
"The name of the coffee pot's model. For instance, Black Delicious."
::= {coffeePot 2}
coffeePotControl  OBJECT-TYPE
SYNTAX  INTEGER {
    on(1),
    off(2)
}
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"This variable controls the current state of the coffee pot's on-off switch. To begin filtering, set it to on(1). To abort filtering (perhaps in the event of an emergency), set it to off(2)."
::= { coffeePot 3 }

coffeePotFillInFilter  OBJECT-TYPE
SYNTAX  INTEGER (0..12)
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"This variable indicates the quantity of ground coffee in the filter. One spoon of ground coffee per cup. 0 indicates no coffee in the filter."
::= { coffeePot 4 }

coffeePotCoffeeType  OBJECT-TYPE
SYNTAX  INTEGER {
    arabica(1),
    brasil(2),
    colombie(3),
    chti-coffee(4),
    other(5)
}
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"This variable gives informations about the type of ground coffee in the filter. Cht'i coffee is coffee with ground chicory. This is the regional custom in the north of France (in 'Cht'i land')."
::= { coffeePot 5 }

coffeePotBistoul  OBJECT-TYPE
SYNTAX  INTEGER {
    no-alcohol(1), -- no health risk
    genievre(2),
    cognac(3),
    armagnac(4),
    calvados(5),
    whisky(6), -- Irish coffee
    other(7)
}
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"The bistoul is another custom of the Cht'i land. You put your coffee and you stir a first time, you put alcohol(*) (genievre or
cognac or calvados...) and you stir your coffee a second time. So as you stir twice ('Bis' in french) this type of coffee is called a bistoul in Ch'ti dialect. In fact in Ch'ti land you don't put sugar in your coffee, you sweet coffee 'a l'chuchette' 'Ti ch't'arconnos t'es d'min coin...'(*) Be careful, alcohol abuse is dangerous for your health."

```::= { coffeePot 6 }

coffeePotTemperature  OBJECT-TYPE
   -- SYNTAX      Gauge
SYNTAX MyGauge
ACCESS read-only
STATUS mandatory
DESCRIPTION
"Coffee temperature."
::= { coffeePot 7 }
```

```coffeePotCounter OBJECT-TYPE
   -- SYNTAX    counter
SYNTAX MyCounter
ACCESS read-only
STATUS mandatory
DESCRIPTION
"Coffee cycles counter, for evaluation of the coffeepot scale level."
::= { coffeePot 8 }
```

END
Notes