# Gerenciamiento de Redes de Datos usando Java & SNMP


**Trabajo Final presentado para obtener el grado de Especialista en Redes y Seguridad**

**Lic. Laura Andrea Fava**

**Director: Javier F. Díaz**

**Facultad de Informática**
**Universidad Nacional de La Plata**

**Septiembre, 2015**

# Indice

**NOMENCLATURAS**

API: Application Programming Interface

ASN: Abstract Syntax Notation

ATM: Asynchronous Transfer Mode

BER: Basic Encoding Rules

INMF: Internet Network Management Framework

ISO/OSI:

JDMK: Java Dynamic Management Kit

JDK: Java Developers Kit

JMAPI: Java Management Application Programming Interface

LAN: Local Area Networks

MIB: Management Information Base

SMDS: Switched Multimegabit Data Service

SNMP: Simple Network Management Protocol

SMI: Structure of Management Information

WAN: Wide Area Network

WLAN: Wireless Local Area Networks

## Contexto

Los años '70 fueron caracterizados por redes de datos centralizadas. Los mainframes fueron los únicos actores de aquella década, caracterizada por bajas velocidades y transmisiones asincrónicas. Las comunicaciones en los '80, fueron afectadas por la llegada de los primeros microprocesadores, los cuales ofrecían mejores precios y velocidades respecto de los mainframes. Con esto, las LANs proliferaron, también surgieron tecnologías para posibilitar transmisiones entre LANs. Las aplicaciones de mainframes comenzaron a migrar a procesamientos más distribuidos, y con esto, surgió la necesidad de un gerenciamiento distribuido.

Hoy, las nuevas tecnologías y la computación distribuida han madurado, las WANs (Wide Area Network) (como ATM –Asynchronous Transfer Mode-, SMDS –Swiyched Multimegabit Data Service- y Frame Relay), están reuniendo las necesidades de las aplicaciones multimediales y de alta velocidad y por último la tecnologías wireless (como WLAN -Wireless Local Area Networks-, BW -Broadband Wireless- y Bluetooth) representan el área de mayor crecimiento en los últimos años. Las herramientas para gerenciamiento de redes también han acompañado esta evolución, apoyadas por tecnologías basadas en web o tecnologías móviles, que utilizan navegadores o *smartphones* para acceder a la información de gerenciamiento.

También es cierto que muchos fabricantes brindan al usuario herramientas para el monitoreo de sus equipos, pero gran parte de las mismas utilizan protocolos propietarios lo que impide la integración y la correlación con la información generada por los equipos de otros fabricantes. Por otro lado, también existen herramientas que facilitan el monitoreo de recursos y se apoyan en protocolos estándares como SNMP, aunque muchas de ellas generan mucho tráfico SNMP, necesitan acceder a la estructura interna de la red y otras, además poseen aplicaciones de gerenciamiento con interfaces de usuario muy textuales y poco amigables.

Este escenario, motivó el estudio del framework de gerenciamiento de redes propuesto por la IETF: funcionamiento, versiones, limitaciones y también el análisis de las APIs disponibles, principalmente las relacionadas con el protocolo SNMP, con el objetivo de proponer alguna herramienta que ayude a los administradores de red en su labor cotidiana de gerenciamiento de redes.

## Objetivos generales y específicos

El objetivo de este trabajo es proponer una arquitectura e implementar un prototipo que facilite el gerenciamiento de redes de datos basadas en SNMP (Simple Network Management Protocol).

Para alcanzar este objetivo se propone:

- Analizar el protocolo SNMP.

- Proponer una arquitectura superadora a la arquitectura SNMP.
- Analizar librerías gratuitas para el desarrollo del prototipo que valide la arquitectura propuesta.
- Implementar un prototipo con las librerías seleccionadas.

## Metodología

Se comenzará a investigar el protocolo SNMP, se realizarán las primeras pruebas a fin de lograr una familiarización con su funcionamiento y con sus mensajes. Se analizarán sus limitaciones.

Se propondrá una arquitectura para gerenciamiento de redes de datos basadas en SNMP.

Se analizarán las APIs disponibles para la implementación de un prototipo que valide la arquitectura propuesta. Se implementará un prototipo de la arquitectura propuesta.

El lenguaje de programación que se utilizará es JAVA, el cual brinda facilidades para efectuar administración vía WEB, desarrollar GUIs de alta calidad para el Gerente, implementar agentes inteligentes para entidades de diferentes vendedores -debido a la independencia de la plataforma-, y contar con una API con funciones especiales para las operaciones de gerenciamiento.

## Introducción

SNMP es un protocolo que ha tenido amplia aceptación porque brinda una interface no propietaria para administrar dispositivos de múltiples vendedores independientemente de sus características y tecnologías de redes. En consecuencia, provee un gerenciamiento a nivel macro, es decir, muchas veces no brinda detalles requeridos para solventar problemas más específicos. Dado que necesita manejar múltiples dispositivos fabricados por diferentes proveedores los parámetros que maneja SNMP son limitados y basados en los estándares definidos. En la actualidad, existen infinidades de dispositivos de redes muy diferentes unos de otros, por lo que muchos aspectos no pueden ser manejados por SNMP.

En este trabajo se analizará el protocolo SNMP, incluyendo la estructura de la información de gerenciamiento (SMI) y las bases de información de gerenciamiento (MIB) que el protocolo maneja, se propone una arquitectura que permita extender las capacidades brindadas por SNMP y se presenta un prototipo de implementación de la propuesta. A continuación se sintetiza la estructura del informe.

Los tres primeros capítulos están destinados a presentar las características del Internet-standard Network Management Framework (INMF). En el capítulo 1 se hace una introducción sobre la sintaxis de la información de gerenciamiento intercambiada entre agentes y gerentes. Para que el agente y el gerente puedan comunicarse deben utilizar la

misma sintaxis para la notación de los datos y también la misma sintaxis para la transferencia o transmisión de los mismos a través de la red. En el capítulo 2 se analizan las MIBs o bases de información de gerenciamiento que describen a los objetos de gerenciamiento en sí mismos. En el capítulo 3, para completar los conceptos que abarca el INMF descriptos en los capítulos anteriores, se describe el protocolo que utilizan los agentes y los gerentes para comunicarse, denominado SNMP de sus siglas en inglés Simple Network Management Protocol.

En el capítulo 4 se describen el estado del arte en cuanto a APIs para SNMP. Se realiza un análisis comparativo entre las mismas y se justifica la elección de *Java Management Extensions* -JMX- para la implementación del prototipo.

En el capítulo 5 se describe la arquitectura propuesta, la cual está destinada a facilitar y mejorar el monitoreo de redes basadas en SNMP. En primera instancia se describe más en detalle que en el caítulo anterior la API para gerenciamiento de redes JMX, la cual provee funcionalidades para implementar agentes que puedan comunicarse con gerentes. Luego se propone una arquitectura basada en JMX que permite extender las capacidades de los agentes SNMP, reducir la carga de la red y favorecer a la administración y seguridad de la misma.

En el capítulo 6 se describen las funcionalidades provistas por el proptotipo teniendo en cuenta las cinco áreas básicas de gerenciemitno del modelo FACPS -del inglés *Fault, Configuration, Accounting, Performance, Security.*

En el capítulo 7 se presentan las conclusiones del presente trabajo, destacándose los aportes y resultados obtenidos y las futuras líneas de investigación.

Finalmente se adjunta un Anexo con secciones de código fuente del prototipo.

# Capítulo 1

## Estructura de la Información de Gerenciamiento (SMI)

### 1.1 Introducción

El IEMF -Internet Estándar Management Framework-, es un framework que incluye el estudio de SNMP –Simple Network Management Protocol-, MIBs –Management Information Base- y de SMI -Structure of Management Information- [Case, J.; RFC1442]. Está basado en una cierta documentación que define tanto la información de gerenciamiento como el protocolo que la administra. En este capítulo se analiza las reglas para definir la estructura de gerenciamiento dadas por SMI y en los próximos capítulos se describen las MIBs y el protocolo SNMP.
En el paradigma agente/gerente para gerenciamiento de redes, los objetos gerenciados deben estar accesibles *física y lógicamente*. Que estén accesibles físicamente significa que los objetos tienen una dirección física a partir de la cual se puede por ejemplo, contar paquetes de entrada/salida o cuantificar de alguna manera la información gerenciada. El acceso lógico se refiere a que la información debe ser almacenada en algún lugar con alguna estructura a partir de la cual pueda ser consultada y modificada.
SMI organiza, nombra y describe la información de gerenciamiento para que la misma pueda ser lógicamente accedida; establece que cada objeto gerenciado debe tener un *nombre*, una *sintaxis*, y una *codificación*. El *nombre* es un dato que caracteriza al objeto y el *OID*, es un número que lo identifica unívocamente, la *sintaxis* define el tipo de dato del objeto y la *codificación* describe cómo es serializada la información asociada con el objeto, para transmitirla entre máquinas.

### 1.2 Información de Gerenciamiento

En términos del Modelo ISO/OSI[1], la sintaxis es una función de capa de presentación -capa 6-, la cual define el formato de la información almacenada dentro de una máquina. Para que el agente y el gerente puedan comunicarse, ambos deben conocer y utilizar la misma sintaxis. Para que esto suceda, dos cosas deben estar estandarizadas: **la sintaxis abstracta** y **la sintaxis para la transferencia**. La primera define especificaciones para

[1] El modelo de interconexión de sistemas abiertos –OSI-, es la propuesta que hizo la Organización Internacional para la Estandarización –ISO- para estandarizar la interconexión de sistemas abiertos.

la notación de los datos y la segunda establece la codificación que deben tener esos datos para la transmisión.

La Internet SMI especifica que **ASN.1[2] define la sintaxis abstracta para mensajes** y **BER[3] provee la sintaxis para la transferencia de datos**. SMI y SNMP usan el lenguaje formal ASN.1 y BER para definir varios aspectos del Internet Network Management Framework.

### 1.2.1 Elementos de la notación de la sintaxis abstracta (ASN.1)

ASN.1 fue diseñado para definir información estructurada -mensajes- en una modalidad independiente de la máquina. Para lograrlo ASN.1 define tipos de datos básicos, tales como enteros y *strings*, y nuevos tipos de datos basados en la combinación de estos tipos básicos y el BER define la forma en que los datos son serializados para la transmisión.
ASN.1 usa algunos términos para definir procedimientos, tipos, macros, asignar valores, los cuales debe escribirse con mayúscula y funcionan como palabras claves. Asegura la comunicación entre las máquinas sin importar su arquitectura interna.

**Tipos y Valores**

Un **Tipo** es una clase de dato, define la estructura del dato que necesita la máquina para comprender y procesar el dato. SMI define 3 tipos de datos:

- Primitivos o simples: INTEGER, OCTET STRING, OBJECT IDENTIFIER, NULL.
- Constructores o agregados: SEQUENCE y SEQUENCE OF, generan listas y tablas.
- Definidos: son los tipos más complejos y descriptivos. Un ejemplo muy común es IpAddress, el cual significa una dirección de Internet de 32 bits.

El **Valor** cuantifica el tipo, provee una instancia específica para dicho tipo. Por ejemplo un valor podría ser el dato de la variable sysDescr o una entrada en una tabla de ruteo.
Es posible definir también, un subconjunto de valores posibles a través de **subtipos**. La especificación del subtipo aparece después que el tipo y muestra el o los valores posibles. Por ejemplo: INTEGER (0…255).

**Macros**

Las macros son notaciones que permiten extender el lenguaje ASN.1. Las definiciones de las MIBs hacen uso extensivo de las macros. La **Figura 1.1** transcribe la definición del

[2] La notación sintáctica abstracta es una norma para representar datos independientemente de la máquina que se esté usando y sus formas de representación internas. Es un protocolo de nivel de presentación en el modelo OSI. El protocolo SNMP usa el ASN.1 para representar sus objetos gestionables. Sus siglas corresponden a *Abstract Syntax Notation.*
[3] Reglas de Codificación que se aplican a la información de gerenciamiento para su transporte sobre la red. Sus siglas BER corresponden a *Basic Encoding Rules.*

primer objeto de la MIB-II, que es la descripción del sistema o **sysDescr** donde puede observarse el uso de la macro ASN.1 **OBJECT_TYPE** para su definición.

```
sysDescr OBJECT-TYPE
          SYNTAX  DisplayString (SIZE (0..255))
          ACCESS  read-only
          STATUS  mandatory
          DESCRIPTION
             "A textual description of the entity.  This value
             should include the full name and version
             identification of the system's hardware type,
             software operating-system, and networking
             software.  It is mandatory that this only contain
             printable ASCII characters."
          ::= { system 1 }
```

**Figura 1.1: Uso de la macro OBJECT-TYPE de SNMP**

Esta macro es una de las más usadas en las definiciones de las MIBs para gerenciamiento porque especifica el identificador del objeto y los datos que están contenidos en la MIB.

### Módulos

ASN.1 permite agrupar descripciones en grupos llamados módulos que pueden formar parte de otras MIBs. Por ejemplo RMON-MIB es un módulo que también es parte de la MIB-II.
Como puede observarse en la **Figura 1.2**, los módulos comienzan con su nombre en mayúsculas, seguidos por DEFINITIONS ::= y el cuerpo encerrado por las sentencias BEGIN y END. Puede contener la sentencia IMPORTS para importar otras definiciones, por ejemplo en el ejemplo citado, la primera línea después del IMPORTS define que el tipo *Counter* corresponde a otro Módulo MIB especificado en la RFC 1155 [Rose, M.; RFC1155]. Por último, el OBJECT-IDENTIFIER identifica que el valor de RMON es el objeto 16 debajo del subárbol de MIB-II.

```
RMON-MIB DEFINITIONS ::= BEGIN
    IMPORTS
              Counter                    RFC-1155-SMI
              DesplayString              RFC-1158-MIB
              mib-2                      RFC-1211-MIB
              OBJECT-TYPE                RFC-1212
              TRAP-TYPE                  RFC-1215;
    -- Remote Network Monitoring Mib
    rmon  OBJECT-IDENTIFIER ::= { mib-2 16 }
     . . .
END
```

**Figura 1.2: Uso de la macro OBJECT-TYPE de SNMP**

ASN.1 define significados especiales para los siguientes símbolos:

| Símbolo | Significado |
|---|---|
| **-** | Número con signo |
| **--** | Comentario |
| **::=** | Asignación |
| **\|** | Opciones de una lista |
| **{}** | Comienzo y fin de una lista |
| **()** | Comienzo y fin de una tag |
| **[]** | Comienzo y fin de un subtipo |
| **..** | Rango |

**Un poco más sobre ASN.1: definición de objetos en las MIBs y tipos de datos**

Una MIB –descripta en detalles en el próximo capítulo- contiene los objetos que serán gerenciados. Como adelantamos existe una macro muy utilizada, **OBJECT-TYPE** que define a esos objetos de una forma estándar consistente para todas las MIBs, tanto públicas como privadas. Además para la definición se pueden usar las siguientes palabras especiales o claves:

- SYNTAX: define la estructura de datos.
- ACCESS: define el acceso permitido a ese objeto. Estos pueden ser read_only, read_writte, not-accesible.
- STATUS: define el soporte de implementación para ese objeto. Los valores pueden ser: mandatory, optional, deprecated, obsoleto. Cuando el STATUS define un valor este tiene alcance para todos los objetos del grupo.
- DESCRIPTION: contiene una definición textual para el objeto.
- INDEX/DEFVAL: se usan únicamente para objetos filas. Indica el lugar en donde el objeto aparece en la fila -o sea la columna-.

La **Figura 1.3** muestra la definición del objeto llamado **`ifNumber`** con un valor de tipo entero que indica la cantidad de interfaces de red del objeto, un dato que se considera sólo de lectura y debe ser

```
ifNumber OBJECT-TYPE
            SYNTAX  INTEGER
            ACCESS  read-only
            STATUS  mandatory
            DESCRIPTION
              "The number of network interfaces regardless
              of their current state) present on this
              system."
            ::= { interfaces 1 }
```

**Figura 1.3: Definición de ifNumber usando ASN.1**

Para mantener la simplicidad de SNMP, SMI usa un subconjunto de los tipos de datos ASN.1 y pueden ser tipos de datos primitivos o tipos constructores.

### (A) Tipos de datos primitivos o simples

**INTEGER** es un tipo primitivo que puede tomar valores enteros positivos y negativos, incluyendo el cero.

```
ipDefaultTTL OBJECT-TYPE
        SYNTAX INTEGER
        ACCESS read-only
        STATUS mandatory
        DESCRIPTION
                "The default value inserted . . ."
        ::= { ip 2}
```

**Figura 1.4: Uso de un tipo simple: INTEGER**

**OCTET STRING** es un tipo primitivo cuyos valores son una secuencia ordenada de ceros, unos o más octetos. SNMP usa 3 casos especiales de este tipo: DisplayString –todos los octetos son caracteres ASCII-, octetBitstring –cadenas de caracteres de más de 32 bits de longitud- y PhysAddress –para representar las direcciones físicas-.

```
sysDescr OBJECT_TYPE
          SYNTAX DisplayString(SIZE(0..255))
          ACCESS read-only
          STATUS mandatory
          DESCRPTION "A textual descrpiton of the entity ......."
::= { system 1}
```

**Figura 1.5: Uso de un tipo simple: *DisplayString***

Notar que el subtipo indica que el tamaño posible para DisplayString está entre 0 y 255 octetos.

**OBJECT IDENTIFIER** es un tipo que nombra o identifica items. Su campo *Value* contiene una lista ordenada de *subidentificadores*. Un ejemplo de este tipo se puede ver el grupo *System* del árbol de objetos MIB-II, cuando asume que un identificador de objeto tiene el siguiente valor:

```
internet OBJECT_IDENTIFIER :: = { iso org(3) dos(6) }
mgmt OBJECT_IDENTIFIER :: = { internet 1 }
sysDescr OBJECT_IDENTIFIER :: = { system 1 }
sysDescr OBJECT_IDENTIFIER :: = { 1.3.6.1.2.1.1.1 }
```

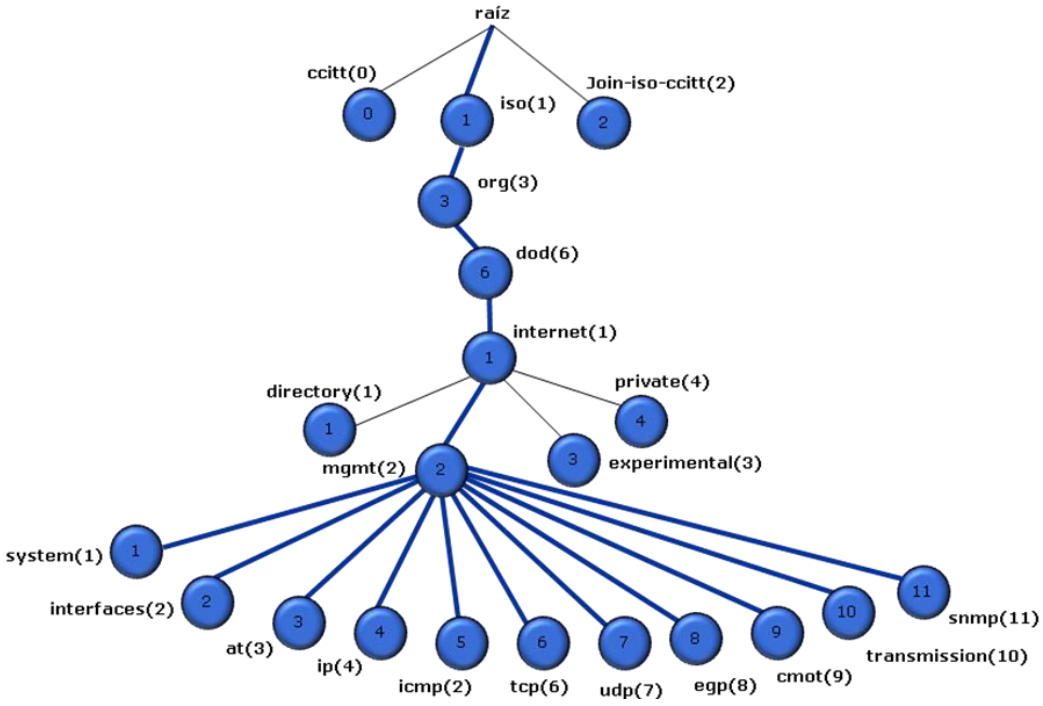



**Figura 1.6: Valores de Identificadores de Objetos asignados a Internet**

**NULL** es un tipo que sirve como relleno, pero no son usados por objetos SNMP.

**(B) Tipos de datos estructurados o Constructores**

Los tipos constructores son **SEQUENCE** y **SEQUENCE OF** y se usan para definir tablas y filas (entradas) dentro de aquellas tablas. Por convención, los nombres de los objetos tablas deben comenzar con *Table* y las filas con *Entry*.

- **SEQUENCE** es un tipo Constructor que define una lista ordenada de tipos. Algunos tipos pueden ser opcionales y todos pueden ser diferentes tipos ASN.1. Puede definir filas en una tabla, donde cada entrada en la **SEQUENCE** especifica una columna dentro de la fila.
- **SEQUENCE OF** es un tipo Constructor que referencia a un tipo simple existente; cada valor en el nuevo tipo es una lista de cero, uno o más valores de aquel tipo existente. Al igual que **SEQUENCE, SEQUENCE OF** puede definir las filas en una tabla; pero a diferencia de ésta, **SEQUENCE OF** sólo usa elementos del mismo tipo ASN.1.

La **Figura 1.7** muestra la definición de la tabla de conexión ***tcp*** donde se observa el uso de ambos tipos:

```
tcpConnTable OBJECT-TYPE
            SYNTAX  SEQUENCE OF TcpConnEntry
            ACCESS  not-accessible
            STATUS  mandatory
            DESCRIPTION
               "A table containing TCP connection-specific
               information."
            ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
           SYNTAX  TcpConnEntry
           ACCESS  not-accessible
           STATUS  mandatory
           DESCRIPTION
                "Information about a particular current TCP
                 connection.  An object of this type is
                 transient, in that it ceases to exist when
                 the connection makes the transition to the
                 CLOSED state."
           INDEX { tcpConnLocalAddress,
                   tcpConnLocalPort,
                   tcpConnRemAddress,
                   tcpConnRemPort }
             ::= { tcpConnTable 1 }

         TcpConnEntry ::=
             SEQUENCE {
                 tcpConnState INTEGER,
                 tcpConnLocalAddress IpAddress,
                 tcpConnLocalPort INTEGER (0..65535),
                 tcpConnRemAddress IpAddress,
                 tcpConnRemPort INTEGER (0..65535)
              }
```

**Figura 1.7: Uso de tipos estructurados**

## Tipos Definidos

El Internet Network Management Framework usa tipos definidos como: IpAddress, NetworkAddress, Counter, Gauge, TimeTicks y Opaque –definidos en la RFC 1155-.
A manera de ejemplo en la **Figura 1.8** se muestra el uso de uno de estos tipos definidos, el **IPAddress**, que representa direcciones de Internet de 32 bits. Este tipo está representado como un OCTET STRING de longitud 4 (octetos).

```
TcpConnRemAddress OBJECT_TYPE
          SYNTAX IPAddress
          ACCESS read-only
          STATUS mandatory
          DESCRPTION
            "The remote IP address for TCP Connection”
::= { tcpConnEntry 4 }
```

**Figura 1.8: Uso de tipos definidos**

### 1.2.2 Reglas de codificación (BER)

Anteriormente se describió como representar la información de gerenciamiento a través de una sintaxis abstracta. Ahora se detallará como codificar esa información para poder transmitirla sobre la red. Las reglas de codificación definen la sintaxis de transferencia y está especificada en la ISO 8825-1[4] [ISO 8825].
Cada máquina en un sistema de gerenciamiento puede tener su propia representación interna de la información de gerenciamiento, pero ASN.1 describe tal información en un formato estándar. La sintaxis para su transferencia ejecuta una comunicación entre máquinas a nivel de bit (representación externa).
Por ejemplo, supongamos que una estación de gerenciamiento necesita información de gerenciamiento desde un dispositivo. La aplicación de gerenciamiento genera un requerimiento SNMP –como veremos en el próximo capítulo- que el BER codifica y transmite sobre la red. El router recibe la información, la decodifica usando las reglas BER e interpreta el comando SNMP. El mecanismo que sucede cuando se retorna es similar, pero de manera reversa. La estructura de la información usada para la representación externa es llamada codificación tipo-longitud-valor. La **Figura 1.9** ejemplifica este proceso.

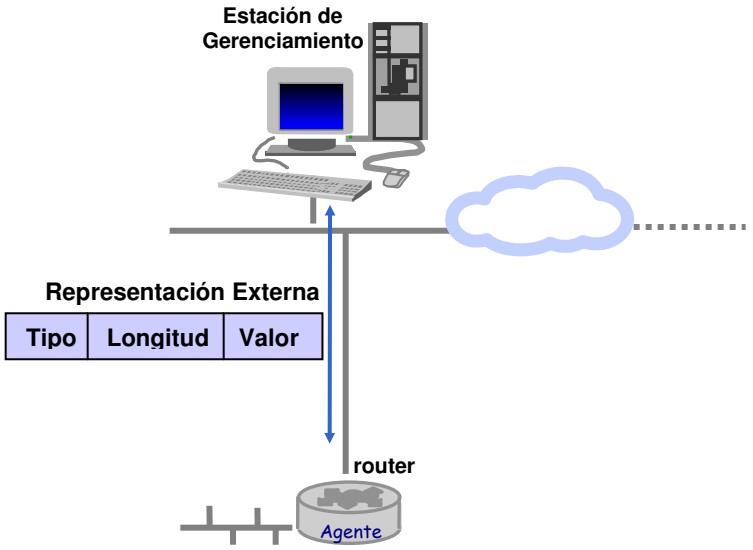


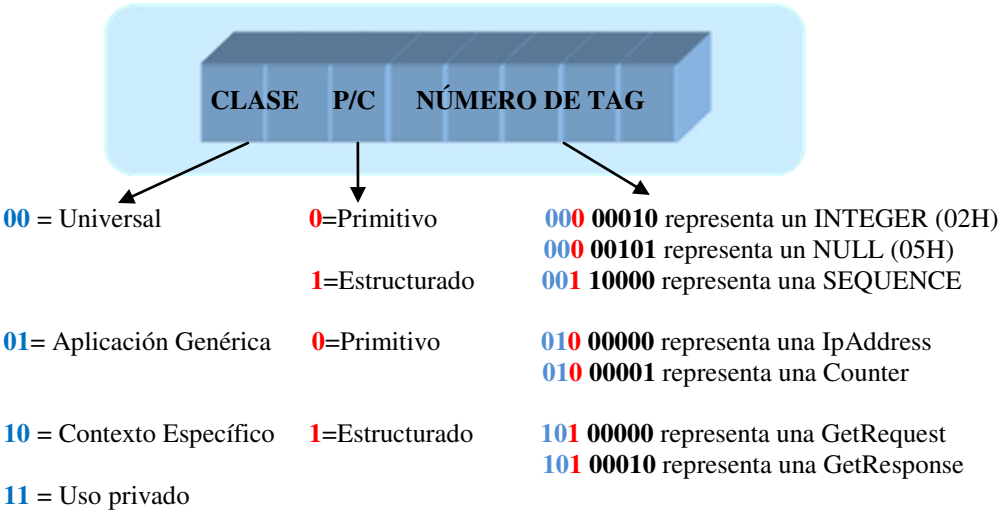

**Figura 1.9: Los mensaje SNMP entre el gerente y los agentes viajan codificados**

Cada dato es codificado -como muestra la figura- como un identificador de **tipo**, una **longitu**d y un **valor** o dato propiamente dicho. Esta codificación también es conocida como codificación TLV –del inglés type-length-value-.

#### El Campo Tipo

[4] Estándar que describe la notación ASN.1.

El tipo va primero y define la estructura de codificación que viene detrás. Contiene una identificación para la estructura: la *clase* y un *número* que indica si el valor es primitivo o estructurado y un *número de tag*.
Como puede observarse en la **Figura 1.10**, este campo tiene "subcampos". El primero de ellos, la *clase (bit 7 y 8)*, es usado por las aplicaciones SNMP para codificar distintos tipos de datos. La clase universal codifica INTEGER, OCTET STIRNG, etc., la clase aplicación codifica tipos definidos como IpAddres, Counter, etc., la clase específica de contexto se usa para codificar unidades de datos SNMP como GetRequest, GetResponse, etc. y finalmente el último tipo es para uso privado.
A la clase le sucede un bit ilustrado *P/C* (bit 6) que indica si la codificación es de un tipo de datos primitivo o estructurado.
Finalmente viene el *número de tag* (bits 1 al 5), que identifica el tipo de dato. Este número, representado en binario está definido por distintos estándares: la ISO 8824-1[5] [ISO 8825] define los números para la clase *Universal*, la especificación SMI RFC 1155 [Rose, M.] define números para la clase *Aplicación* y la especificación SNMP RFC 1157 [Case, J.] contiene números para la clase *Contexto Específico*.



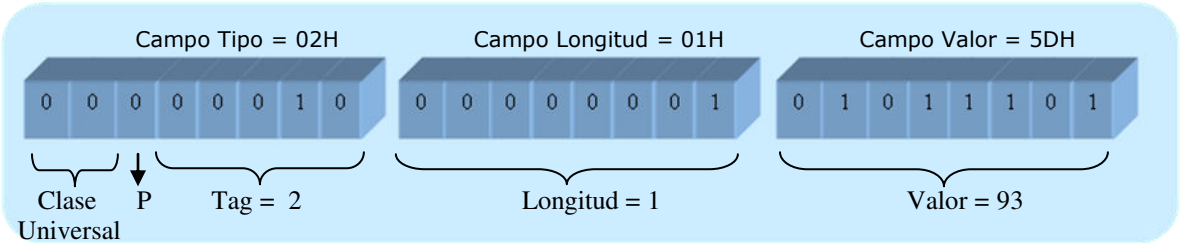

**La Figura 1.10 muestra la estructura interna del campo Tipo**

### Los campo Longitud y Valor

El campo *longitud* está a continuación del campo tipo y determina la cantidad de octetos que contendrá el campo valor. Puede indicar una longitud determinada o puede indicar que esta indefinido, esto es, no se conoce la longitud del dato antes de su transmisión. El campo *valor* contiene cero o más octetos con el contenido. A continuación describo algunos datos codificados donde puede verse la estructura de codificación TVL.

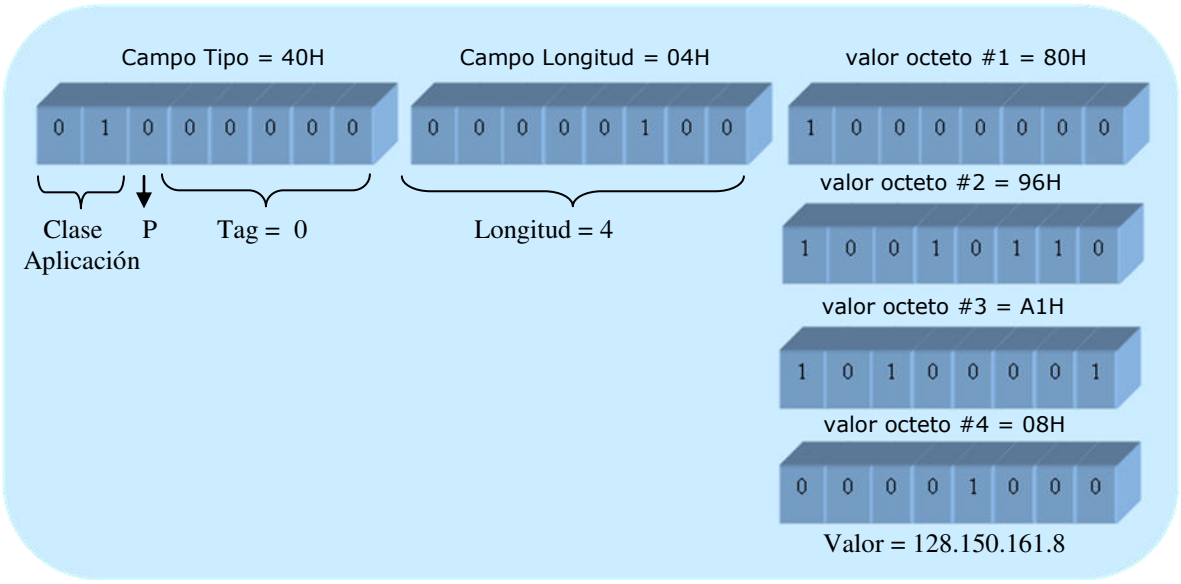
[5] Estándar que describe las reglas de codificación de ASN.1.

La **Figura 1.11** muestra la codificación para un tipo **INTEGER**, correspondiente a un tipo primitivo universal, que puede contener como valor uno o más octetos.



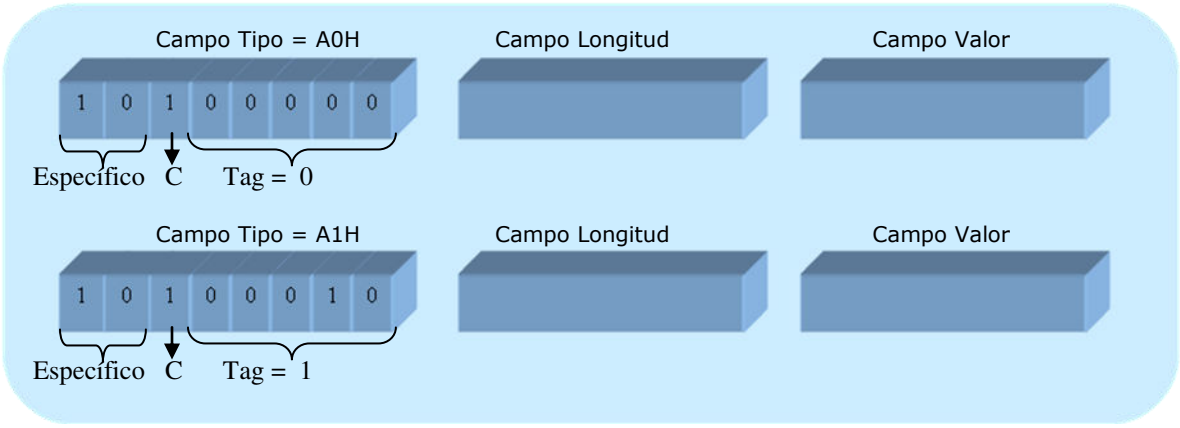

**La Figura 1.11 Codificación para el tipo INTEGER, valor = "93"**

La **Figura 1.12** muestra la codificación para un tipo **IpAddres**, un tipo Primitivo de clase Aplicación.




**La Figura 1.12 Codificación para el tipo IpAddress, valor = "128.150.161.8"**

Finalmente, la **Figura 1.13** ejemplifica la codificación para tipos específicos del contexto, particularmente los usados dentro del contexto de SNMP, de interés para esta tesis. Como veremos en el capítulo 3, SNMP provee 5 unidades de datos del protocolo SNMP o PDU –protocol data unit- o mensajes a saber: GetRequest, GetNextRequest, GetResponse, SetResponse y Trap. Los PDU tienen número de tag de 0 a 4 respectivamente. La imagen muestra la codificación para el GetRequest y para el GetNextRequest. Los campos Longitud y Valor dependen de la información transportada.




**La Figura 1.13 Codificación para tipos específicos del contexto usados en SNMP, *getResquest* y *getResponse***

## Síntesis

En este capítulo se ha analizado la sintaxis de la información de gerenciamiento intercambiada entre agentes y gerentes. Para que el agente y el gerente puedan comunicarse deben utilizar la misma sintaxis para la notación de los datos y también la misma sintaxis para la transferencia o transmisión de los mismos a través de la red. Como se ha descripto, la Internet SMI especifica que ASN.1 define la sintaxis abstracta para mensajes y BER provee la sintaxis para la transferencia de datos. SMI y SNMP usan el lenguaje formal ASN.1 y BER para definir varios aspectos del Internet Network Management Framework.

En el próximo capítulo se analizan las MIBs o bases de información de gerenciamiento que describen a los objetos de gerenciamiento en sí mismos y en el capítulo subsiguiente, para completar los conceptos que abarca el Internet Network Management Framework, se describe el protocolo que utilizan los agentes y los gerentes para comunicarse, denominado SNMP de sus siglas en inglés Simple Network Management Protocol.

# Capítulo 2

## Información de Gerenciamiento:
## ¿qué se intercambia en las conversaciones de gerenciamiento?

### 2.1 Introducción

En el capítulo anterior se ha descripto SMI, una sintaxis para organizar, describir y recuperar información de gerenciamiento y se ha analizado la codificación que se utiliza para el transporte de dicha información. En este capítulo se examinará la documentación estándar que se utiliza para definir MIBs.

### 2.2 Estructura de las MIBs

Una MIB es una definición precisa de la información que está disponible a través de un protocolo de gerenciamiento de red. Cada MIB usa la estructura jerárquica definida en ASN.1 para organizar toda la información disponible; las MIBs son definiciones formales ASN.1 que mantienen la información de gerenciamiento.

La **Figura 3.1** muestra la parte superior del árbol MIB estándar según la definición de la ISO 8824-1 [ISO 8824] con una raíz sin numeración y con 3 subárboles: *ccitt(0)*, *iso(1)* y *Join-iso-ccitt(2)*. Cada elemento del árbol es un *nodo etiquetado* y cada nodo contiene un *identificador* y una *descripción*. Los elementos que son hojas, representan un dispositivo o una característica de un dispositivo y son unívocamente identificados por tal identificador –secuencia de enteros separados por "." que es el resultado del camino desde la raíz hasta la hoja-.

En la **Figura 3.1** la sintaxis, iso(1), indica que este nodo tiene un número identificador uno ("1") en ese nivel del árbol. Una rama importante para esta tesis es la formada por el nodo iso(1), luego el nodo org(3) y debajo de éste un nodo usado por el Departamento de Defensa de los Estados Unidos, dod(6) y debajo internet(1). Toda la información obtenida desde dispositivos de internet a través de los protocolos

DOD[1], tales como TCP/IP, se ubican en este subárbol ***internet(1)*** que tiene el identificador de objeto ***1.3.6.1***.

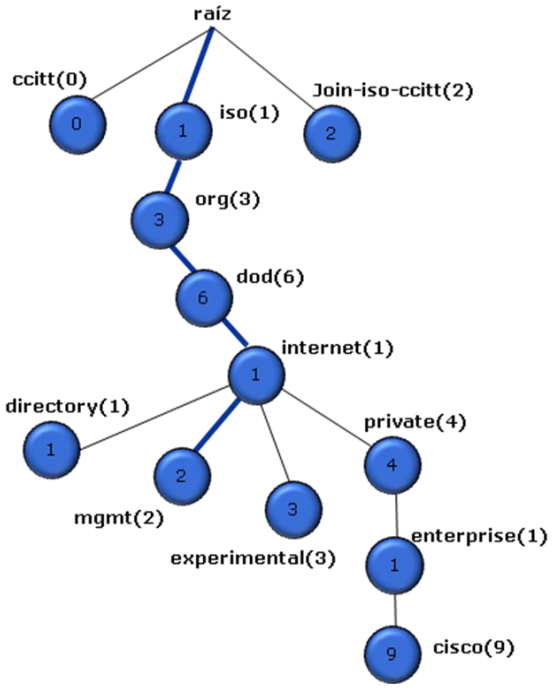



**Figura 3.1: Estructura del árbol MIB con una extensión de un subárbol privado (CISCO)**

Debajo de internet(1), tienden cuatro sub-árboles:

- **directory(1):** este subárbol contiene información sobre el servicio de directorios OSI X.500
- **mgmt(2):** es utilizado para gerenciamiento de los protocolos DOD. Los objetos de este subárbol, son usados para obtener información específica de los dispositivos y están ampliamente implementados. Estos objetos son divididos en 11 categorías ilustradas en la **Figura 3.2**.
- **experimental(3)**, usados para ubicar los protocolos experimentales y MIB en desarrollo.
- **private(4)**, es usado para especificar objetos definidos unilateralmente. Cada subárbol de este nodo es asignado a una simple empresa, la empresa entonces puede crear atributos bajo este subárbol específicos para sus productos. Las MIBs específicas del vendedor, son implementadas para complementar las MIBs estándares y proveer la funcionalidad necesaria para el gerenciamiento de la red, incluyendo las características de productos específicos del vendedor.

[1] DoD o DOD, Departamento de Defensa de los Estados Unidos, pionero en la elaboración de estándares para Internet.

## 2.3 Grupos definidos en la MIB-I y MIB-II

La **Figura 3.2** grafica los grupos definidos en las MIBs estándares para Internet, MIB-I definida en la RFC 1156 [Rose, M.; RFC1156] y MIB-II definida en la RFC 1213 [McCloghrie, K.].

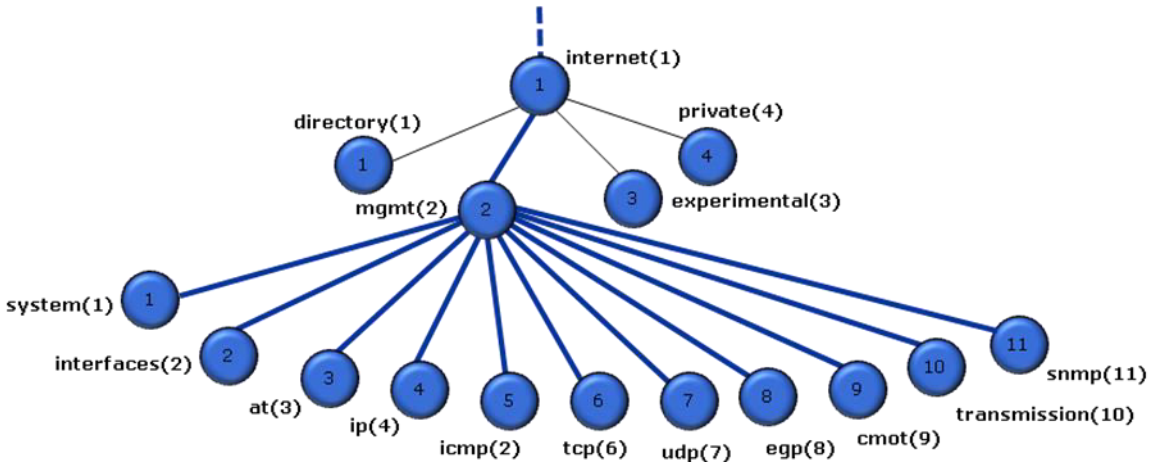



**Figura 3.2: Estructura del subárbol mgmt(2)**

Los objetos gerenciados son agrupados en grupos por 2 razones:

- el agrupamiento lógico facilita el uso de identificadores de objetos –OID-. Por ejemplo, un identificador de objeto con un prefijo {1.3.6.1.2.1} indica que este es un objeto definido dentro de MIB-I o MIB-II y un identificador de objeto con un prefijo {1.3.6.1.2.1.4} indica que estamos hablando de un objeto definido por una empresa privada.
- diseño de agentes SNMP es más directo. La implementación de un grupo implica la implementación de todos los objetos dentro del grupo. Así, tanto el programador como el usuario final, es de comprensión directa cuando se habla que una MIB tiene soporte para un grupo determinado, por ejemplo para el grupo TCP.

La siguiente tabla sintetiza la información provista por cada grupo y algunos de sus atributos:

| Categoría | Información provista | Atributos/Objetos?? |
|---|---|---|
| **system (1)** | Provee una descripción textual de una entidad. | `sysContact, sysLocation, sysDescr, sysName.` |
| **interfaces (2)** | Provee información sobre las interfaces de hardware del dispositivo gerenciado. Esta información está representada como una tabla. | `Para cada interface: ifDescr, ifType, ifSpeed, ifPhysicalAddress, ifOperStatus, etc.` |
| **address translation (3)** | Brinda mapeo de direcciones IP a direcciones físicas. Ésta información también está en forma de tabla. | `Para cada dirección: atPhysAddress, atNetAddress` |
| **ip (4)** | Provee información del protocolo de | `ipForwarding, ipDefaultTTL, ipFragOKs,` |

| | | |
|---|---|---|
| | Internet, principalmente de estadísticas de datagramas relacionados con el protocolo IP. | `ipFragFails, ipAddrTable, etc.` |
| **icmp (5)** | Contiene información de mensajes de control y representa varias operaciones ICMP dentro de la entidad gerenciada. | `icmpInMsgs, icmpInErrors, icmpInEchos, icmpInEchosReps, icmpOutEchos, icmpOutEchosReps, etc.` |
| **tcp (6)** | Este grupo contiene información sobre operaciones y conexiones del Protocolo de Transporte, TCP. | `tcpRtoMin, tcpRtoMax, tcpMaxConn, tcpInErrors, etc.` |
| **udp (7)** | Contiene información sobre operaciones del Protocolo de Datagramas de Usuario, UDP. | `udpInDatagrams, udpNoPorts, udpLocalAddress, etc.` |
| **egp (8)** | Este grupo contiene información sobre operaciones del Protocolo de Gateway exterior, EGP. | `egpNeighAs, egpNeighInMsgs, egpNeighOutMsgs, etc.` |
| **cmot (9)** | Servicios de Información de gerenciamiento comunes s/ TCP. | La RFC 1213 no define explícitamente ninguno de estos objetos. |
| **transmission (10)** | Contiene objetos relacionados con la transmisión de datos. | La RFC 1213 no define explícitamente ninguno de estos objetos. |
| **snmp (11)** | Provee información sobre objetos SNMP. | `snmpInPkts, snmpOutPkts, snmpInGetRequests, snmpInGetNexts, etc.` |

Con la necesidad creciente de gerenciamiento de redes en general, se han desarrollado una gran cantidad de MIBs, que soportan arquitecturas, protocolos y medios de transmisión específicos. Algunas de las MIBs específicas incluídas bajo el árbol MIB-II son: *OSPF* (1.3.6.1.2.1.14, RFC 1253 [Coltun, R.]), *BGP* (1.3.6.1.2.1.15, RFC 1657 [Willis, S.]), *RMON* (1.3.6.1.2.1.16, RFC 1757 [Waldbusser, S.]), *ATM Objects* (1.3.6.1.2.1.37, RFC 1695 [Ahmed, M.]), entre otras.

## 2.4 Los grupos de la MIB-II

En este punto se detallan algunos de los objetos que pertenecen a los grupos de MIB-II, que son utilizados para configuración, manejo de fallas y en especial aquellos utilizados para manejo de performance.

### 2.4.1 El grupo SYSTEM

Este grupo contiene información acerca del sistema en donde reside el agente SNMP. Estos objetos son útiles para tareas de configuración y manejo de fallas. Por ejemplo, los objetos ***sysName, SysLocation, sysContact*** son útiles para la configuración de una red e indican el nombre del dispositivo, la ubicación y el contacto. Este grupo también

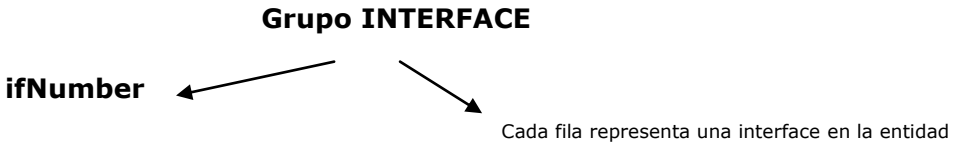
provee atributos para el manejo de fallas como el objeto ***sysUpTime*** el cual indica cuanto tiempo hace que el sistema está operativo.

### 2.4.2 El grupo INTERFACE

El grupo INTERFACE contiene una tabla con información sobre todas las interfaces de red disponibles en el dispositivo. Para cada una de ellas se utiliza una instancia de este objeto y son seleccionadas por un número de índice.

Estos objetos brindan de este grupo contienen información acerca de cada interface de los dispositivos de la red, útil para manejo de fallas, configuración y performance.

La **Figura 3.3** ilustra la estructura de este grupo, correspondiente a un dispositivo con 3 interfaces.




| ifIndex | ifDescr | ifType | ifMtu | ifSpeed | ifPhysAddress | . |
|---|---|---|---|---|---|---|
| **1** | FastEthernet0/1 | ethernetCsmacd | 1500 | 1200000 | 00:15:1a:67:00:8:0 | |
| **2** | FastEthernet0/2 | ethernetCsmacd | 1500 | 1100000 | 00:50:73:43:bb:f0 | |
| **3** | Loopback0 | softwareLoopback | 1515 | 4294960 | 00:4f:4e:00:13:c9 | |

**Figura 3.3: Estructura del grupo INTERFACE**

El primer objeto ***ifNumber*** indica el número de interfaces que dispone el dispositivo y para cada interface existe una fila en la tabla. Hay objetos como ***ifName, ifType, ifSpeed, ifMTU,*** *que* son útiles para la configuración de cada una de las interfaces de los dispositivos de una red y los objetos como ***ifAdminStatus*** y ***ifOperStatus*** que permiten determinar el estado (`up`, `down`, `testing`, `unknow`) de las interfaces y cuanto tiempo hace que el sistema está operativo.

#### *Objetos usados para manejar Performance*

Los objetos de este grupo permiten calcular por ejemplo, la cantidad de paquetes recibidos y enviados y los porcentajes de error de paquetes entrantes y salientes.

Los cálculos para obtener estos parámetros se harían de la siguiente manera:

***Total de paquetes recibidos = ifInUcastPkts + ifInBroadcastsPkts + ifInMulticatsPkts***

***Total de paquetes enviados = ifOutUcastPkts + ifOutBroadcastsPkts + ifOutMulticatsPkts***

***Porcentaje de error en paquetes entrantes = ifInErrors/Total de paquetes recibidos***

***Porcentaje de error en paquetes salientes = ifOutErrors/Total de paquetes enviados***

Otro dato importante para determinar la performance de una interface es calcular el porcentaje de utilización de la misma. Para esto se debe calcular el total de *bytes* por segundo y dividirlo por la velocidad:

***Bytes por segundo = ((ifInOctetsx-IfInOctetsy)+(ifOutOctetsx-IfOutOctetsy))/(x-y)***

***Utilización = (Bytes por segundo*8)/ifSpeed***

Otro objeto de este grupo que puede usarse para manejar performance, es ***ifOutQLen*** que indica si un dispositivo está teniendo problemas enviando datos. Este valor crece cuando la cantidad de paquetes esperando por salir –dejar la interface- aumenta. Si es persistente este crecimiento podría indicar congestión en la interface. El objeto ***ifOutQLen*** usado junto con ***ifOutDiscards***-cantidad de paquetes que se descartan porque no pueden dejar la interface- y con ***ifOutOctets***-cantidad de bytes que salen- puede dar una indicación de posible congestión en la red. Si ***ifOutDiscards*** *crece y* ***ifOutOctets*** decrece, podría indicar que la interface esta congestionada.

Estos son solo algunos de los objetos y fórmulas que pueden usarse para manejar la performance de la red.

### 2.4.3 El grupo IP

Los objetos de este grupo contienen información acerca del protocolo IP[2] en una entidad. Esta información va desde objetos que dan información sobre errores o direcciones IP en una entidad hasta tablas complejas de ruteo y de mapeo.

La estructura del grupo IP está ilustrada en la **Figura 3.4**, donde pueden observarse cuatro áreas.

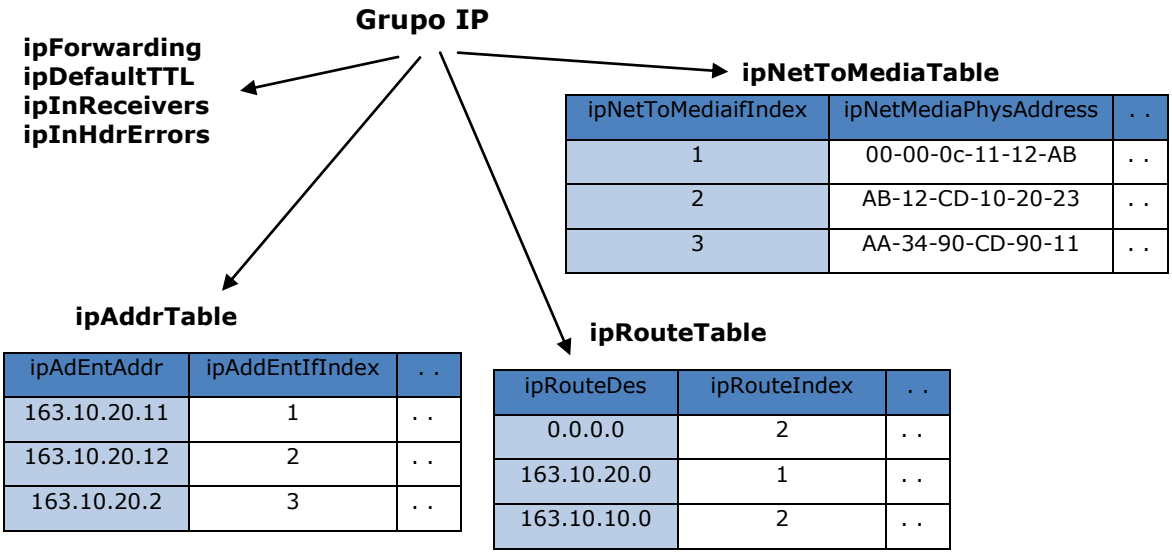



**Figura 3.4: Estructura del grupo IP**

[2] IP es un protocolo estándar de red que utiliza para entrega de paquetes con una modalidad no orientada a conexión.

### *Objetos usados para manejar Performance*

Este grupo tiene una gran cantidad de objetos para manejar la performance. Por ejemplo para medir el porcentaje de tráfico IP y el porcentaje de errores de paquetes IP podríamos usar las siguientes fórmulas con atributos del grupo IP:

***Porcentaje de paquetes IP recibidos =***

***IpInReceives/(ifInUcastPkts+ifInMulticastPkts+ifInBroadcastPkts)***

***Porcentaje de paquetes IP enviados =***

***IpOutRequests/(ifOutUcastPkts + ifOutMulticastPkts + ifOutBroadcastPkts)***

En estas fórmulas, *ipInReceives* indica el total de paquetes recibidos por la entidad y *ipOutRequests* cuenta el número de datagramas IP enviados por la entidad.

**Porcentaje de errores en IP entrantes =**

**(*ipInDiscards* + *ipInHdrErrors* + *ipInAddrErrors*) /*ipInReceives***

**Porcentaje de errores en IP salientes =**

**(*ipOutDiscards* + *ipOutHdrErrors* + *ipOutAddrErrors*) /*ipOutRequests***

Otros objetos interesantes son el ***ipRoutingDiscards*** que indica si la entidad está descartando datagramas IP válidos por carencia de recursos -este valor puede indicar si la entidad cuenta o no con los recursos necesarios como para proveer alta performance a red- y el ***InUnknownProtos*** que indica la cantidad de datagramas que la entidad descarta porque no soporta el protocolo -si este valor es alto, puede afectar la performance de la red porque la entidad consume recursos innecesarios chequeando si tiene errores y determinando si la dirección IP es local o no, y luego analiza si soporta el protocolo-.

## 2.4.4 El grupo ICMP

El grupo ICMP[3] contiene objetos que dan información sobre el protocolo ICMP [Ahmed, M.] en la entidad.

### *Objetos usados para manejar Performance*

La gran mayoría de los objetos de este grupo son usados para performance. Cuando una entidad soporta este grupo, debe procesar cada paquete ICMP recibido haciendo que esto pueda afectar negativamente la performance de la entidad.

Los atributos de este grupo nos permiten recuperar el porcentaje de paquetes ICMP respecto del total de paquetes manejados para analizar la causa de su existencia.

Para calcular el porcentaje de paquetes ICMP primero se debe obtener la cantidad de paquetes recibido y enviados y a esta suma dividirla por ***icmpInMsgs + icmpOutMsgs***. Que una entidad reciba o envíe una gran cantidad de paquetes ICMP no significa que tenga problemas de performance, pero estas estadísticas pueden ayudar a descubrir problemas.

[3] ICMP es un protocolo estándar que transporta mensajes de control y de error para dispositivos IP

Los objetos de este grupo también muestran la cantidad de cada tipo de paquetes ICMP. Por ejemplo una entidad recibiendo un gran número de paquetes ***icmnInRedirects*** podría indicar problemas de tablas de ruteo y performance en la red.

### 2.4.5 El grupo TCP

Los objetos del grupo TCP[4] pueden ayudar para manejar la configuración y a medir la performance de una entidad. Como es un protocolo de capa de transporte, este protocolo trata con control de flujo, congestión y retransmisión de segmentos perdidos. Al igual que el grupo IP, hay objetos generales que dan información acerca de TCP y una tabla valores de cada conexión TCP.

La estructura del grupo TCP está ilustrada en la **Figura 3.5**

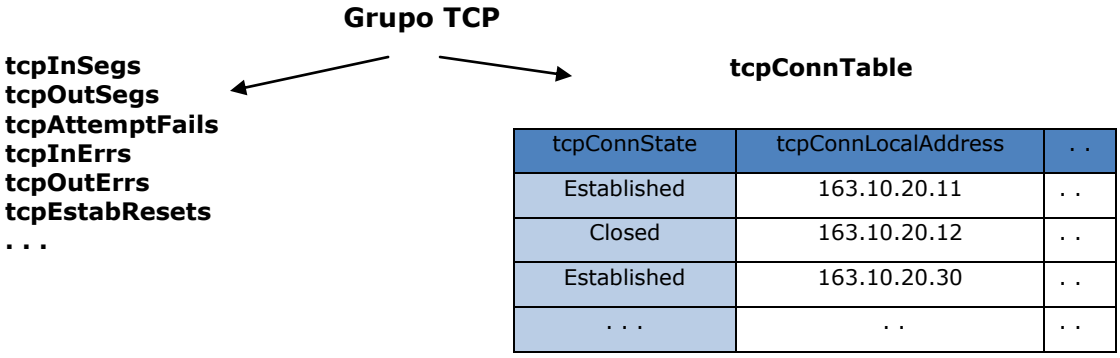



**Figura 3.5: Estructura del grupo TCP**

#### *Objetos usados para manejar Performance*

Un intento por establecer una conexión TCP puede fallar por una cantidad de razones: el destino no existe, la red tiene una falla. Este grupo tiene objetos como ***tcpAttemptFails*** y ***tcpEstabResets*** que brindan información sobre la cantidad de intentos fallidos y permiten medir la tasa de rechazos de conexión. El objeto ***tcpRetransSegs*** nos da el número de segmentos que son re-enviados en la red. La retransmisión de segmentos no necesariamente refleja un problema de performance, sin embargo, el número de retransmisiones puede indicar si la entidad está enviando múltiples copias en un esfuerzo por asegurar confiabilidad.

### 2.4.6 El grupo UDP

El grupo UDP[5] provee un conjunto limitado de objetos que dan información acerca del protocolo UDP en la entidad y para aplicaciones UDP en la entidad. La estructura del grupo UDP está ilustrada en la **Figura 3.6**

[4] TCP es un protocolo de transporte estándar que provee conexiones confiables entre aplicaciones.
[5] UDP es un protocolo no orientado a conexión, simple y no confiable de capa de transporte.

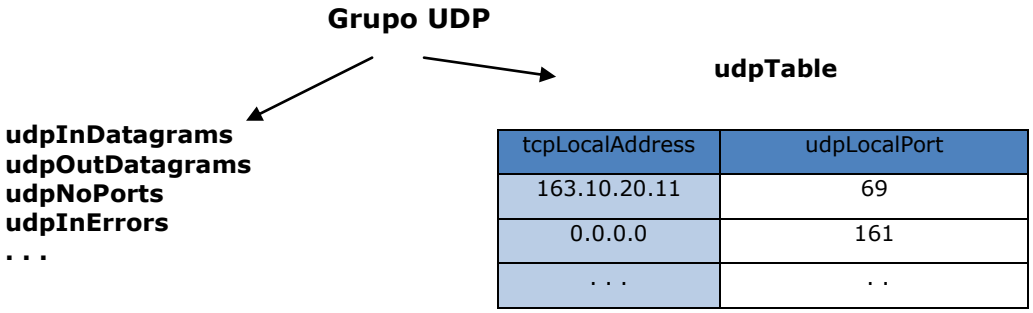



**Figura 3.6: Estructura del grupo UDP**

Como UDP no establece conexiones, las tablas no tienen información acerca de las conexiones actuales –pues nunca existen-, en realidad la tabla muestra los puertos locales y las direcciones de redes relacionadas.

### *Objetos usados para manejar Performance*

Al igual que el grupo TCP, podemos obtener la tasa de paquetes UDP con los objetos ***udpInDatagrams*** y ***udpOutDatagrams*** en un período de tiempo.

Este grupo también tiene el objeto ***udpInErrors*** que puede informar sobre errores específicos en la red. Como UDP es un protocolo simple, sólo hace CRC[6] o *chequeos de redundancia cíclica* en la parte de datos del datagrama. Un datagrama UDP puede contener errores CRC por varios motivos, como error en un link o en un software.

SNMP usa UDP como protocolo de transporte. Si un sistema para gerenciamiento de redes está teniendo problemas recibiendo datagramas UDP desde el sistema remoto, el contador local ***udpInErrors*** podría indicar que los datagramas conteniendo la información SNMP posiblemente no esté haciéndolo exitosamente a través de la red.

## 2.4.7 El grupo EGP

Las redes IP pueden estar agrupadas en áreas lógicas llamadas *sistemas autónomos*. Un sistema autónomo comúnmente es una red y sus subredes asociadas o un conjunto de redes y subredes bajo una misma administración. Dos dispositivos de red pertenecientes a diferentes sistemas autónomos pueden determinar su accesibilidad a través del protocolo EGP[7]. Los dispositivos de red que se comunican entre sistemas autónomos usando EGP son llamados *vecinos EGP*.

La estructura del grupo EGP está ilustrada en la **Figura 3.7** y al igual que otros grupos provee un conjunto limitado de objetos que dan información acerca del protocolo EGP en la entidad y tiene una tabla con información de los vecinos EGP. Cada proceso EGP tiene una relación uno a uno con cada vecino.

[6] CRC–cyclic redundancy checks-, http://en.wikipedia.org/wiki/Cyclic_redundancy_check.

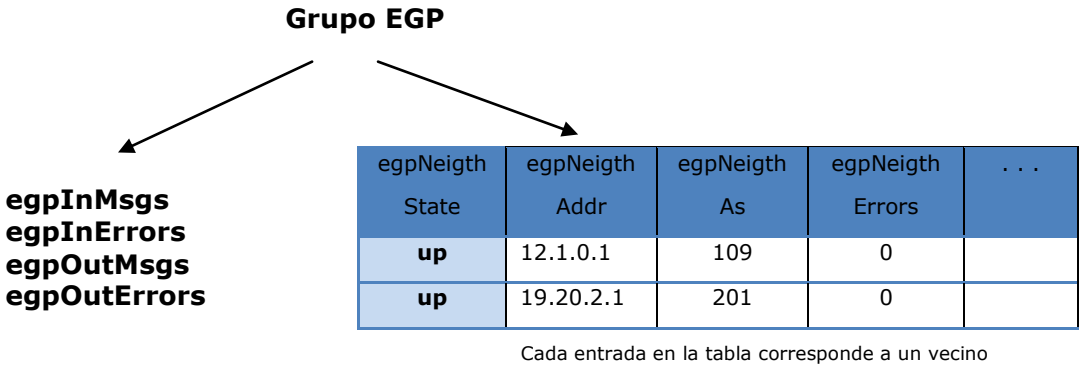



**Figura 3.7: Estructura del grupo EGP**

### 2.4.8 El grupo SNMP

Este grupo da información acerca de paquetes y errores entrantes y salientes del protocolo SNMP en la entidad. Hay un total de 30 objetos SNMP que nos permiten por ejemplo medir -a través de los objetos ***snmpInPkts*** y ***snmpOutPkts***- la cantidad de recursos que una entidad está usando para manejar SNMP. Una alta tasa de estos objetos podría sugerir que el manager está actualmente requiriendo información de la entidad.

También brinda objetos para hacer manejo de fallas como ***snmpInASNParseErrs*** que da información sobre el número total de errores ASN.1 y VER encontrados por el protocolo SNMP en la entidad cuando decodifica mensajes SNMP recibidos.

## Síntesis

Este capítulo se da una definición precisa de MIBs y se analiza cada uno de los grupos que componen el subárbol de gerenciamiento internet {1.3.6.1}. Se muestran algunos ejemplos de uso de los objetos que componen los grupos para gerenciamiento de redes en internet. En los ejemplos de uso se puso énfasis en el área de *performance*, sin dejar de lado a los objetos que permiten administrar la *configuración*, *manejo de fallas*, *seguridad* y *accounting.*

Si bien el propósito de esta tesis no es enseñar a hacer gerenciamiento sino definir una arquitectura que permita y facilite el gerenciamiento de datos, para poder definir un prototipo que se corresponda con tal arquitectura, es necesario comprender la constitución de estos grupos y las tareas que desarrollan los administradores en su labor diaria. En el próximo capítulo se detalla el protocolo SNMP, el cual permite inspeccionar y modificar la información de gerenciamiento para un elemento de red.

[7] EGP es un protocolo que le indica a un dispositivo de red IP acerca de la accesibilidad de otra red IP.

# Capítulo 3

## Simple Network Managemnt Protocolo (SNMP)

### 3.1 Introducción

En los capítulos anteriores se han detallado la estructura de la información de gerenciamiento (SMI) y las bases de información de gerenciamiento (MIB). Ahora para completar los conceptos que abarca el Internet Network Management Framework, se describirá el protocolo Simple Network Management Protocol [Case, J.; RFC1157].
Los primeros métodos usados para la recolección de datos consistieron en pequeños *softwares* propietarios, los cuales eran instalados por los propios proveedores a medida que lanzaban sus productos. Por este motivo, para manejar 2 dispositivos con la misma funcionalidad, pero de distintos proveedores, a menudo se debía aprender a manejar diferentes métodos para la recolección de información.
Otra de las posibilidades que tenían los administradores era utilizar el protocolo de Internet -IP-, el cual provee un mecanismo de mensajes de propósito especial, el ICMP, con el cual podían juntar información útil para gerenciar la red. Cualquier dispositivo con el paquete IP completo, posee los mensajes *Echo y Echo Reply*, que si bien no son fáciles de interpretar por cualquier persona, conforman un método simple y rápido para un administrador.
Usando estos mensajes, un host que recibe un mensaje *Echo*, debe retornar un mensaje ICMP *Echo Replay*, al host fuente. La ausencia de este último indica la falta de conectividad. Hay una aplicación llamada *ping -Packet InterNet Groper-*, que hace esto. La **Figura 3.1** ilustra este mecanismo.

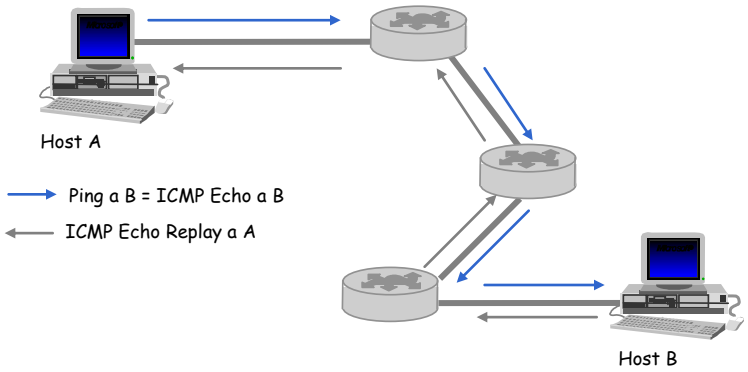



**Figura 3.1: Ping usado para testear conectividad en la red.**

TCP/IP no es el único paquete de protocolos que posee el ping, también lo incluyen otros como Appletalk, Novel/IPX, Xerox XNS, etc. A pesar de su utilidad, este mecanismo presenta algunas desventajas:

- Las entregas de mensajes no son confiables, por lo que la falta del Echo Replay no siempre representa pérdida de conectividad -la causa podría una congestión-.
- Se necesita *poolings* constantes.
- La información provista es limitada.

Es así como comienzan a surgir nuevos protocolos con el objetivo de acceder de manera uniforme a dispositivos de diversos proveedores. Por el año 1988 surgen los protocolos Hegh-level Entity Managment System -HEMS- que no llegó a nada, Common Management Information Protocol -CMIP- que fue implementado sobre TCP/IP y cosntituyó el CMOT y Simple Gateway Monitoring Protocol -SGMP- que fue la base para el desarrollo del *SNMP*v1. Posteriormente en el año 1992 se desarrolló *SNMPv2* [Case, J.; RFC1906] con el objetivo de mejorar algunas falencias del SNMP, sin embargo, ambas versiones del SNMP carecieron de seguridad, especialmente relacionada con autenticación y privacidad. Un último conjunto de RFCs, conocidas como *SNMPv3*, corrigieron estas deficiencias[1].

Este capítulo comienza con una descripción de los fundamentos del protocolo SNMPv1, comunes para todas las versiones y finaliza el con la descripción de las versiones posteriores.

## 3.2. Arquitectura y funcionamiento de SNMP

Según la RFC 1157, la arquitectura SNMP está compuesta por una colección de *estaciones de gerenciamiento de red* y *elementos de red*. Las estaciones de gerenciamiento ejecutan aplicaciones de gerenciamiento las cuales monitorean y controlan los elementos de red. Los elementos de red son dispositivos que tienen agentes responsables de ejecutar tareas de gerenciamiento requeridas por las aplicaciones de gerenciamiento. Por último, hay un protocolo estándar encargado de comunicar información de gerenciamiento entre las estaciones y los elementos de red.

La **Figura 3.2** ilustra el modelo agente/gerente de SNMP con una estación de gerenciamiento o *manager* SNMP y dos elementos de red –un router y un host- con sus respectivos agentes SNMP.

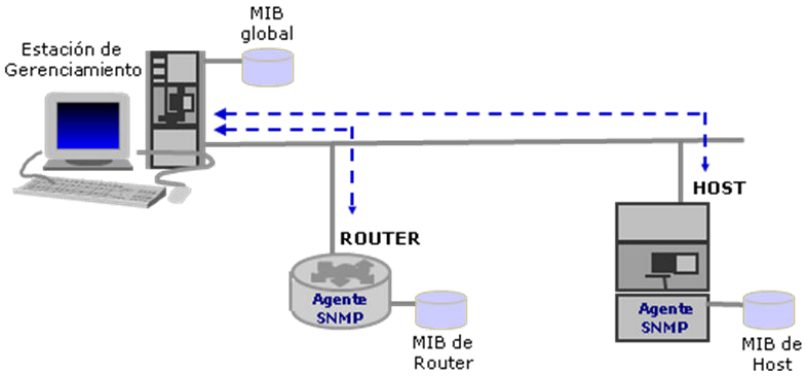



**Figura 3.2: Modelo agente/gerente SNMP**

1 El protocolo SNMPv3 está especificado en las RFCs 3410 hasta 3418, siendola RFC 3416 la principal porque especifica el protocolo en sí mismo.

Un *agente SNMP* es capaz de responder consultas válidas provenientes de una *estación SNMP* o de setear parámetros de configuración en un dispositivo. Tanto la estación de gerenciamiento como cada elemento de red o dispositivo tienen asociados una MIB. Como analizamos en los capítulos anteriores las MIBs tienen una organización lógica que responde a SMI. Ésta define una estructura jerárquica que comienza con una raíz y dispone de ramas que organizan a los objetos gerenciados en categorías lógicas.
Como muestra la imagen, las MIBs representan a los objetos gerenciados con valores en las hojas de sus ramas. Por otro lado es preciso destacar que la MIB del router es diferente de la del host. Primero porque esos dispositivos provienen de diferentes proveedores y segundo porque brindan diferentes funcionalidades adentro de la red y podrían manejan diferente información. Por ejemplo un *host* podría no necesitar tabla de ruteo y un *router* podría no disponer de información sobre uso de CPU que es tan significante para un *host*.
Finalmente, un protocolo de gerenciamiento permite que se comuniquen la aplicación de gerenciamiento y los agentes. Para organizar esta comunicación, el protocolo define mensajes específicos conocidos como comandos, respuestas y notificaciones. Los bloques de construcción de estos mensajes se llaman unidades de datos de protocolo o ***PDUs – protocol data units-.***

Para que nos sirva como base teórica para este capítulo la **Figura 3.3** muestra las capas del modelo ISO/OSI -a partir del cual se desarrollaron los protocolos de Internet TCP/IP- y la arquitectura por niveles de SNMP.

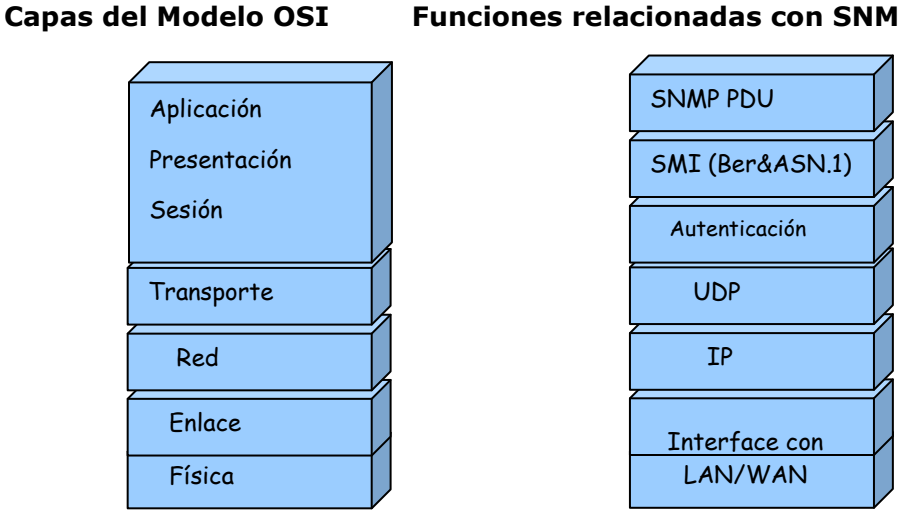



**Figura 3.3: Comparación del Modelo OSI con el Modelo SNMP**

Analicemos un ejemplo para ver cómo interactúan las funciones entre las capas de la arquitectura SNMP y para mostrar una cierta correspondencia con las funcionalidades de las capas del modelo OSI. Supongamos que un manager hace un requerimiento SNMP a un agente, el ASN.1 codificando la capa de aplicación provee la sintaxis apropiada para el mensaje. Las funciones restantes autentican los datos y comunican el requerimiento. El

canal de comunicación entre el gerente y los agentes es sin conexión debido a que la información de gerenciamiento no es tan crítica y no necesita una comunicación confiable. Si comparamos ambos modelos, la comunicación sin conexión del SNMP evita la necesidad de la capa de sesión y reduce responsabilidades de las cuatro capas de abajo. Para la mayoría de las implementaciones el User Datagram Protocol (UDP) desempeña las funciones de la capa de transporte, el protocolo de Internet (IP) provee las funciones de la capa de red, y las LANs (como Ethernet o Token Ring) o WANs (como Frame Relay) proveen las funciones de la capa física y de enlace.

Esta comparación nos provee una base teórica que nos sirve para todo el capítulo. Desde una perspectiva más práctica la **Figura 3.4** ilustra cómo trabaja el modelo SNMP en las capas del modelo OSI, donde participan algunos elementos ya presentados como la aplicación de gerenciamiento, el agente y los mensajes SNMP que comunican la información de gerenciamiento vía cinco unidades de datos del protocolo (PDUs) SNMP. La aplicación de gerenciamiento genera los mensajes *Get, GetNext* o *Set* y el agente responde con el mensaje *GetResponse* o también puede generar un *Trap* cuando sucede una situación particular –algunas veces llamado evento-.

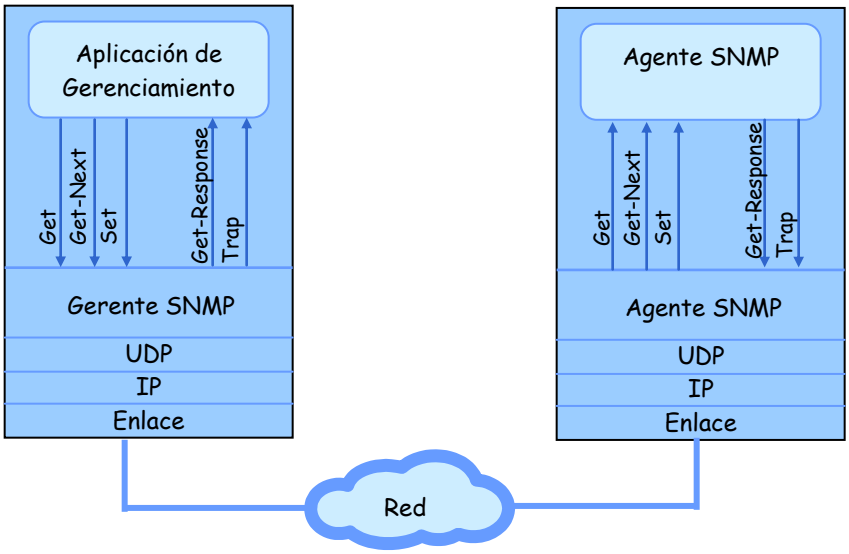



**Figura 3.4: Funcionamiento de SNMP en las capas del Modelo**

Cuando una entidad que participa del protocolo SNMP *genera un mensaje* ejecuta las siguientes acciones:

- Construye el PDU apropiado (por ejemplo, GetRequest-PDU) como un objeto ASN.1.
- Luego pasa este objeto ASN.1 con un nombre de comunidad, su dirección IP origen y destino, al servicio que implementa el esquema de autenticación deseada. Este servicio de autenticación, retorna otro servicio ASN.1.
- Luego la entidad construye un mensaje ASN.1 usando el nombre de la comunidad, resultando en un nuevo objeto ASN.1.

- Este nuevo objeto ASN.1 es *serializado*, usando las reglas de codificación básicas y es enviado usando un servicio de transporte.

Las acciones que ejecuta el protocolo en la entidad que *recibe un mensaje* son similares:

- Ejecuta un *parsing* del datagrama entrante para construir un objeto ASN.1 correspondiente a un mensaje ASN.1. Si esta operación falla, se descarta el datagrama.
- Luego verifica la versión del mensaje. Si no corresponde, se descarta el datagrama.
- La entidad pasa el nombre de la comunidad y los datos del usuario encontrados en el mensaje ASN.1 con las direcciones IP al servicio para que implemente el esquema de autenticación. La entidad retorna otro objeto ASN.1.
- La entidad ejecuta un rudimentario *parsing* sobre este objeto para construir un nuevo objeto ASN.1 para ser procesado. Si como resultado de este proceso se debe retornar un datagrama, entonces la dirección IP origen pasa a ser la de destino y se envía la respuesta.

**Identificando instancias de Objetos**

El protocolo SNMP provee las operaciones que son usadas para accede a la MIB e interactuar con ella. Todas esas operaciones utilizan OIDs para referenciar a los objetos en la MIB, por esto, una comprensión básica del concepto de OIDs es un prerrequisito para comprender SNMP. A continuación se describe los identificadores de objetos y se ejemplifican los casos más comunes.
Como ya analizamos en los capítulos anteriores, los nombres de todos los tipos de objeto en la MIB se definen explícitamente ya sea en el estándar de Internet MIB o en otros documentos que se ajustan a las convenciones de nomenclatura del SMI. A su vez, existen mecanismos para identificar instancias individuales de esos tipos de objetos.
Cada instancia de cualquier tipo de objeto definido en el MIB se identifica con un nombre único llamado *nombre de variable –variable name-*. En general, el nombre de una variable SNMP es un OBJECT IDENTIFIER (OID) de la forma *x.y*, donde *x* es el nombre de un tipo de objeto definido en el MIB –prefijo- y el *y* es un identificador de instancia -postfijo-.

El protocolo SNMP provee acceso a dos tipos de variables:
- *variables escalares*: contienen valores simples como strings, enteros, counters gauges, etc.
- *variables tabulares*: mantiene información sobre entidades similares pertenecientes a un dispositivo. Esas entidades podrían ser interfaces de un *router* o un *switch*, máquinas virtuales en un servidor, etc. La información de cada entidad es representada por una fila, cuyas columnas son variables -escalares- que mantienen la información acerca de la entidad. Una fila es llamada una entrada "entry" en una MIB; cada columna tiene un OID prefijo más un índice único que especifica una fila particular.

Analizamos primero la identificación de un objeto de tipo escalar, **`sysDescr`** y luego uno de tipo columnar, **`ifTable`**, ambos derivados de la RFC 1157, sección 3.2.6.3.

La **Figura 3.5** muestra el tipo de objeto **`sysDescr`**. Para obtener el valor de **`sysDescr`** de un objeto al prefijo del *nombre de variable* **`{1.3.6.1.2.1.1.1}`** se le debe agregar un sub-identificador 0.

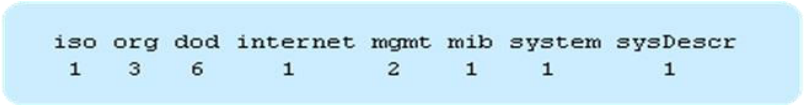



**Figura 3.5: OID de sysDescr**

➔ Resulta entonces que **`{1.3.6.1.2.1.1.1.0}`** identifica a la única instancia de este objeto.

Otro ejemplo un poco más complejo es el objeto **`ifTable`**. La **Figura 3.6** muestra esta clase de objeto que podría tener más de una instancia como resultado de una consulta.

Supongamos que se quiere obtener el valor del objeto **`ifPhysAddress`** –la dirección MAC- de un dispositivo. Si analizamos el árbol desde arriba, el OID de este objeto es **`{1.3.6.1.2.1.2.2.1.6}`** resultado del grupo **`interfaces{1.3.6.1.2.1.2}`**, la **`ifATable{2}`** y el **`ifEntry(1)`**.

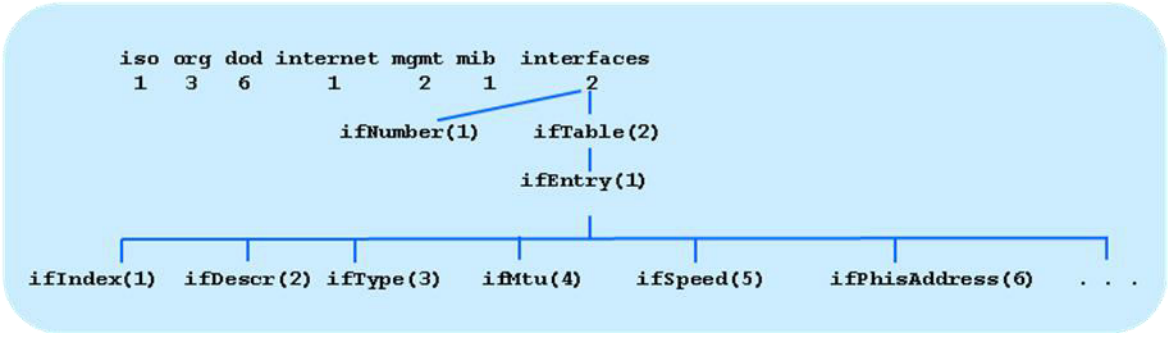



**Figura 3.6: OID de ifTable**

El **`ifTable`** se compone de una secuencia de **`ifEntrys`** que forman las filas de la tabla. Cada fila (cada **`ifEntry`**) tiene un único índice y una serie de variables empezando por **`ifIndex`** como muestra la **Figura 3.7**. Cada índice representa el número de *port* de la interface y los valores individuales son representados por el nombre de la columna, seguido de su índice.

| ifIndex | ifDescr | ifType | ifMtu | ifSpeed | ifPhysAddress | . |
|---|---|---|---|---|---|---|
| **1** | FastEthernet0/1 | ethernetCsmacd | 1500 | 1200000 | 00:15:1a:67:00:8:0 | |
| **2** | FastEthernet0/2 | ethernetCsmacd | 1500 | 1100000 | 00:50:73:43:bb:f0 | |
| **3** | Loopback0 | softwareLoopback | 1515 | 4294960 | 00:4f:4e:00:13:c9 | |

**Figura 3.7: `ifTable{1.3.6.1.2.1.2.2.1}` con valores de ejemplo**

➔ Resulta entonces **`ifSpeed.3`** (o el OID **`1.3.6.1.2.1.2.2.1.3`**) representaría a la variable **`ifSpeed`** para la fila 3 de la tabla.

Después de haber analizado los mecanismos para identificar a las instancias de los objetos, discutiremos los tipos de mensajes que brinda SNMP para transportar los requerimientos y respuestas entre manager y agentes. Los PDUs que se describirán a continuación utilizan las instancias de los objetos para brindar la información requerida por el manager.

## 3.3 Estructura de las PDUs SNMP

El protocolo de gerenciamiento de red es un protocolo de capa de aplicación por el cual las variables de la MIB de un agente pueden ser inspeccionadas o alteradas. La comunicación entre las entidades que participan del protocolo es acompañada por el intercambio de mensajes, cada uno de los cuales es representado independientemente dentro de un datagrama UDP, usando las reglas de codificación básicas del ASN.1.
Una entidad que participa del protocolo SNMP -*entidad protocolo*- recibe mensajes UDP en el port 161 sobre el host con el cual está asociado para todos los mensajes, excepto para aquellos que reportan traps que son recibidos sobre el port 162.
La **Figura 3.8** ilustra un mensaje SNMP encapsulado en un datagrama UDP, con un Header que contiene: port fuente, port destino, longitud y checksum. El datagrama UDP a su vez está encapsulado en un datagrama IP, que a su vez está contenido en un *frame* que viaja por la red local. Cuando el datagrama IP es demasiado grande, puede ser dividido en varios *frames* para la transmisión sobre la LAN.

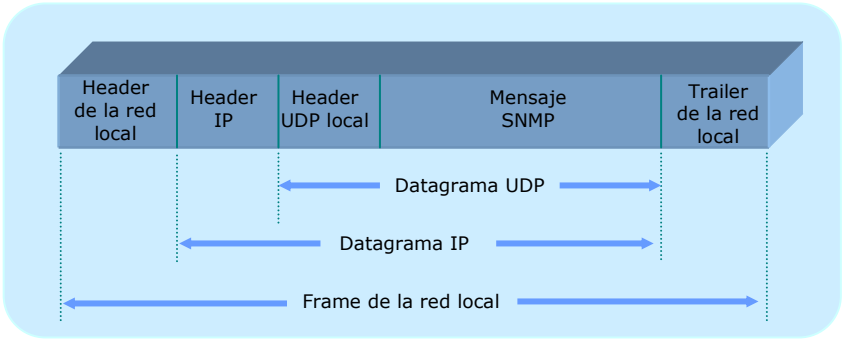



**Figura 3.8: Mensaje SNMP dentro del un *frame* de transmisión local**

Las operaciones SNMP son comunicadas entre managers y agentes usando ***mensajes SNMP***, los cuales constan de tres secciones, un identificador de versión, la *community –o password-* y el PDU del tipo correspondiente. Las dos primeras secciones algunas veces son referenciadas como *header de autenticación SNMP* y la tercera sección está destinada a uno de los 5 mensajes SNMP posibles: GetRequest-PDU, GetNextRequest-PDU, GetResponse-PDU, SetRequest-PDU y trap_PDU. La **Figura 3.10** ilustra la estructura de estos mensajes.


**Figura 3.10: Estructura del mensaje SNMP**

El campo ***version*** asegura que el manager y el agente estén manejando la misma versión del protocolo SNMP. Los mensajes entre el manager y el agente de versiones distintas, son descartados sin ser procesados.
La ***community*** es un string usado por el agente para autenticar al manager antes de permitirle el acceso a su MIB. La *community* junto con la dirección IP del manager son almacenados en un archivo en el agente. Si el número de versión y la *community* que vienen desde el manager coinciden con alguno de los almacenados en este archivo del agente, comienza el procesamiento del mensaje SNMP, de lo contrario el agente informa el error –enviando un Trap-PDU como se describe luego-.

***- Los mensajes Getters, Setters y Responses***

Los mensajes GetRequest, GetNextRequest, SetRequest y GetResponse comparten un mismo formato PDU, mientras que el formato del Trap es diferente y se describirá más adelante en este capítulo. La **Figura 3.10** muestra el formato de estos cuatro mensajes.

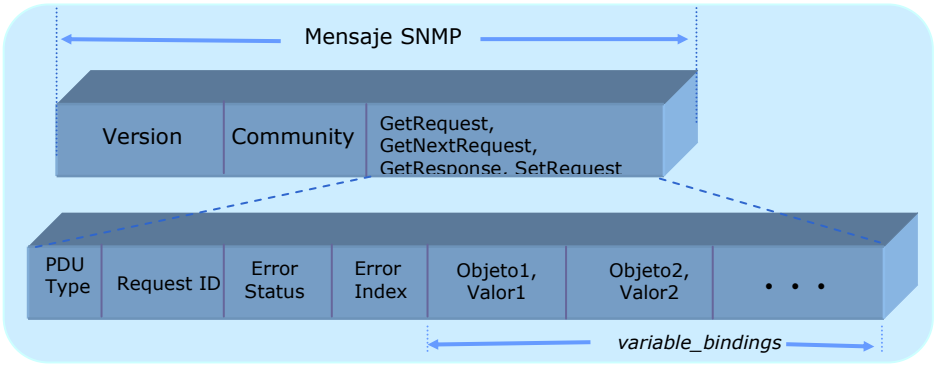



**Figura 3.10: Estructura de los mensajes Getters, Setters y Responses**

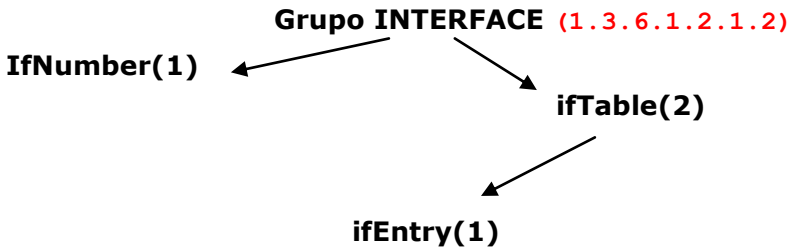
*El formato del PDU -y el mensaje en sí mismo- es formalmente especificado en ASN.1 y codificado en strings que son enviados usando reglas de codificación básicas o BER.*
El PDU ilustrado en la **Figura 3.10** comienza con el campo, *PDU Type* que especifica el *tipo de PDU* que contiene el mensaje. La siguiente tabla muestra los números asignados para cada tipo:

| Tipo de mensaje | Valor del campo |
|---|---|
| GetRequest | 0 |
| GetNextRequest | 1 |
| GetResponse | 2 |
| SetRequest | 3 |

El campo *Request ID* es un número INTEGER que relaciona el requerimiento del gerente con la respuesta del agente. Le sigue el campo *Error Status, también un INTEGER,* que indica si la operación terminó normalmente -noError- o si ocurrió alguno de los 5 tipos que se detallan a continuación.

| Error | Valor del Campo | Significado |
|---|---|---|
| noError | 0 | Operación agente/gerente apropiada |
| tooBig | 1 | El tamaño del *GetRequest* excede a la limitación local |
| noSuchName | 2 | El objeto requerido no coincide con ninguno de los de la MIB |
| badValue | 3 | Un *SetRequest* contiene un valor erróneo para la variable |
| readOnly | 4 | No se define su uso en la RFC 1157 |
| genErr | 5 | Otros errores no definidos explícitamente |

El campo siguiente es *Error Index* el cual identifica la posición dentro de la lista Variables Bindings que causó el error. Por ejemplo si ocurre un error BadValue, entonces el Error Index será igual a 3.
Por último la lista v*ariables_bindings* es una lista de pares de nombre de la variable (la codificación OID del tipo de objeto más la instancia) y valor para la misma (que es NULL para el caso de GetRequest y GetNextRequest).
La aplicación de gerenciamiento utiliza el PDU Get para recuperar el valor de uno o más objetos de la MIB del agente. Tanto el ***GetRequest*** como el ***GetNextRequest*** son generados por una aplicación de gerenciamiento SNMP. En la mayoría de los casos el primero se utiliza para recuperar un valor escalar y el segundo para recuperar valores desde una tabla. El ***GetRequest*** permite recuperar información de gerenciamiento desde un agente. Este mensaje contiene una lista de variables –*variable_binding*- que identifican a los objetos que se están solicitando. Consideremos la MIB de la **Figura 3.7** para hacer una consulta usando el comando ***GetRequest***.




| ifIndex (1) | ifDescr (2) | ifType (3) | ifMtu( 4) | ifSpeed (5) | ifPhysAddress (6) | . |
|---|---|---|---|---|---|---|
| **1** | FastEthernet0/1 | ethernetCsmacd | 1500 | 1200000 | 00:15:1a:67:00:8:0 | |
| **2** | FastEthernet0/2 | ethernetCsmacd_64 | 1500 | 1100000 | 00:50:73:43:bb:f0 | |
| **3** | Loopback0 | softwareLoopback | 1515 | 4294960 | 00:4f:4e:00:13:c9 | |

**Figura 3.7: `ifTable{1.3.6.1.2.1.2.2.1}` con valores de ejemplo**

```
PDU type: GET
Object identifier 1: 1.3.6.1.2.1.2.1 (IF-MIB::ifNumber) => retorna 3
```

El mensaje ***GetRequest*** puede llevar una lista de OIDs y recuperar más de un objeto a la vez.
Un ***GetNextRequest*** se utiliza para recuperar información de gerenciamiento desde un agente, tal como lo hace un ***GetRequest***, sin embargo, los OIDs de la *variable_bindings* no especifican exactamente el OID del objeto a recuperar. En su lugar, para cada OID especificado en el requerimiento, lo que se le solicita al agente es que retorne el objeto que le sigue a ese OID en el orden lexicográfico –el elemento de su derecha-.

```
PDU type: GET-NEXT
Object identifier 1: 1.3.6.1.2.1.2.2.1.3 (IF-MIB::ifType)
```

Retorna el valor del **`ifType`** de la Fila 1 **`=> ethernetCsmacd`**

```
PDU type: GET-NEXT
Object identifier 1: 1.3.6.1.2.1.2.2.1.3.1 (IF-MIB::ifType)
```

Retorna el valor del **`ifType`** de la Fila 2 **`=> ethernetCsmacd_64`**

```
PDU type: GET-NEXT
Object identifier 1: 1.3.6.1.2.1.2.2.1.3.2 (IF-MIB::ifType)
```

Retorna el valor del **`ifType`** de la Fila 2 **`=> softwareLoopback`**

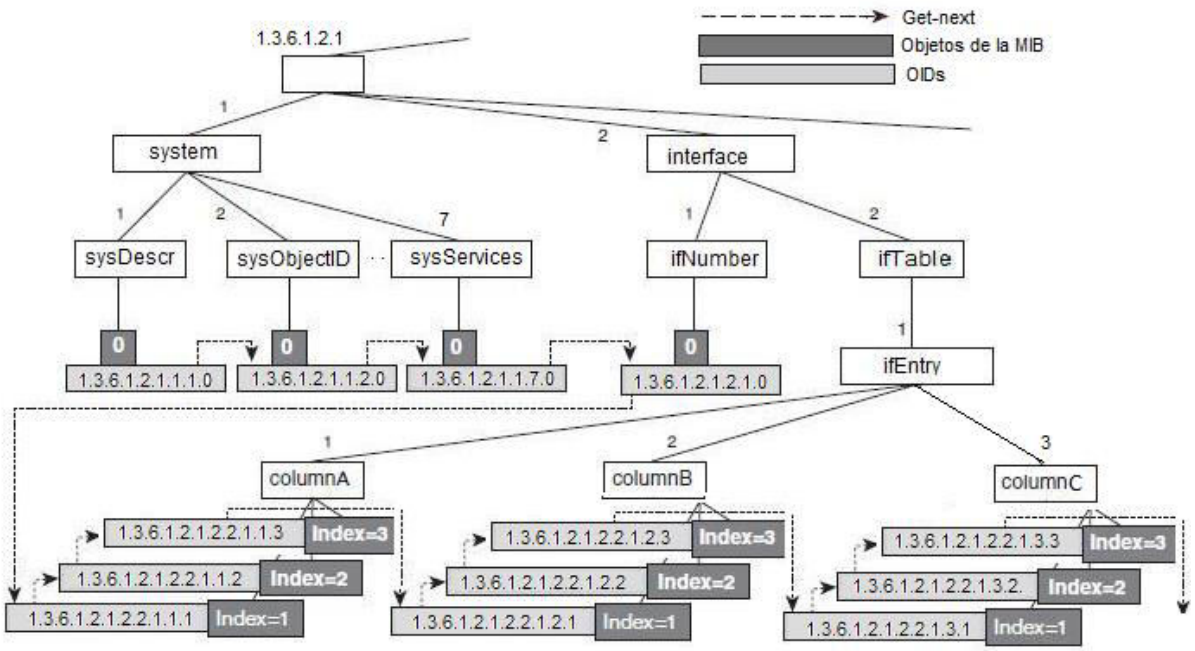



**Figura 3.8: Navegación de la MIB con GetNextRequest**

Un ***GetNextRequest*** que especifica el OID de un "tipo de objeto" es equivalente a un ***GetNextRequest*** con el OID correspondiente al primer objeto de ese tipo –el objeto con el índice menor en el caso de un objeto columnar o 0 en el caso escalar-.

Para comprender la navegación de la MIB usando ***GetNextRequest***, en la **Figura 3.8** se muestra la navegación de la rama INTERFACE que venimos trabajando.
La aplicación de gerenciamiento también es la encargada de generar el ***SetRequest*** para asignar un valor particular a un objeto en la MIB del agente. Si bien la estructura del PDU es igual a la de los get, en este caso, la variable-bindings lleva además de los OIDs los valores que se debe setear para cada OID.

Finalmente el ***GetResponse*** es generado por un agente SNMP después de recibido un GetRequest, GetNextRequest o SetRequest. Un ***GetResponse*** incluye en la respuesta un identificador del requerimiento, un *error status* que contiene un código que indica si el requerimiento se pudo responder en forma exitosa o falló, *error index* con un número de error y la *variable_bindings* con los OID y valores recuperados.

***- Los mensajes Trap***

Un trap SNMP es un mensaje SNMP no solicitado que un agente envía a una gerente. Estos mensajes informan a la entidad aplicación la ocurrencia de algún evento y su estructura interna esta graficada en la **Figura 3.11**.
El primer campo, *PDU Type*, corresponde al tipo de PDU que siempre lleva el valor 4. Luego está el campo *enterprise* que es el nombre de la empresa que lo define, y le sigue la dirección IP del agente que contiene la dirección IP del dispositivo donde ejecuta el agente.

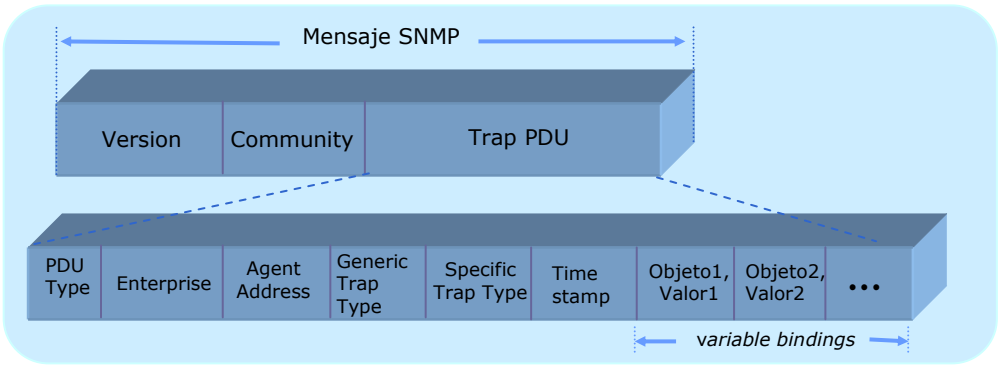



**Figura 3.11: Estructura del mensaje Trap-PDU**

Detrás de la dirección se encuentra el campo *Generic trap* que provee información específica sobre el evento que se está reportando. Existen siete valores predefinidos para este campo:

- El trap **coldStart, valor 0**: significa que la entidad protocolo que envía está reinicializándose, lo que implica que la configuración del agente o la implementación de la entidad protocolo podrían alterarse. Por ejemplo, este trap ocurre cuando se enciende un dispositivo.
- El trap **warmStart, valor 1:** significa que la entidad protocolo que envía está reinicializándose, pero ninguna configuración de agente, ni la implementación de la entidad protocolo serán alteradas. Por ejemplo, este trap ocurre cuando se presiona CTRL+ALT+DEL en un host. Este evento generalmente ocurre por intervención manual.

- El trap **linkDown, valor 2**: indica que la entidad protocolo que envía reconoce una falla en uno de los enlaces representados en la configuración del agente. Este tipo de PDU contiene como primer elemento de su *variable-bindings*, el nombre y el valor de la instancia ifIndex para la interface afectada.
- El trap **linkUp, valor 3**: significa que la entidad protocolo que envía reconoce que uno de los enlaces de comunicación representados en la configuración del agente se levantó recientemente. Este tipo de PDU contiene como primer elemento de su *variable-bindings*, el nombre y el valor de la instancia ifIndex para la interface afectada.
- El trap **authenticationFailure, valor 4**: indica que la entidad protocolo que envía es el destinatario de un mensaje de protocolo, que no ha sido autenticado correctamente. Por ejemplo, si un agente recibe un requerimiento con una *community string* errónea.
- El trap **egpNeighborLoss, valor 5**: significa que un vecino EGP ha sido marcado como caído y no se puede contactar. Este tipo de PDU contiene como primer elemento de su *variable-bindings*, el nombre y el valor de la instancia egpNeighAddr, para el vecino afectado.
- El trap **enterpriseSpecific, valor 6**: significa que la entidad protocolo que envía reconoce que algún evento propietario ha ocurrido.

Los *specific-trap* son implementados por los proveedores para brindar algún servicio adicional que complemente los traps genéricos. Por ejemplo, algunas compañías han implementado traps para informar que el espacio de un disco se está agotando, que el uso de CPU es muy alto, etc. Dos campos adicionales completan el trap PDU, el *timestamp* que indica el tiempo transcurrido desde que comenzó a funcionar el agente. El último campo contiene la *varibale-binding* que ya describimos.

La primera versión del protocolo tuvo tres importantes problemas:
- Es un estándar únicamente para redes de datos IP.
- Es ineficiente para recuperar datos largos.
- Tiene mecanismos de seguridad completamente violables, la *community string*, el único mecanismo de seguridad de esta versión es texto legible –ASCII-.

Estas desventajas dieron lugar a la nueva versión del SNMPv2 y posteriormente SNMPv3 para solucionar algunas de estas falencias.

*Si bien SNMPv1 ganó un amplio apoyo, algunos aspectos de él resultaron demasiado simples. SNMPv1 es notoriamente ineficiente en la recuperación de gran cantidad de información de gerencia -no conoce sobre requerimientos a granel- y ofrece una mínima seguridad que lo hace vulnerable a ciertas amenazas, resultando no recomendable su uso para cambiar la configuración de los dispositivos gestionados -en muchos casos el riesgo de comprometer la integridad de la red es simplemente demasiado grande-. Por este motivo SNMPv1 se terminó utilizando principalmente para monitoreo, a pesar de haber sido originalmente concebido como un protocolo genérico para cubrir todas las funciones de gestión. Otra deficiencia está relacionada con la expresividad del lenguaje de especificación, SMI, y la falta de capacidades tales como la creación y destrucción de entidades lógicas de una manera más simple por parte de las aplicaciones de gerenciamiento.*

## 3.4 Simple Network Management Protocol version 2 (SNMPv2)

Por las razones descriptas en el párrafo anterior, se introdujo *SNMPv2*, una segunda versión de SNMP para solucionar las limitaciones y problemas más urgentes. El aspecto más importante de este protocolo fue la introducción de dos nuevas operaciones de gestión, además de las ya conocidas del SNMPv1: el ***getBulkRequest*** y el ***informRequest***.

SNMPv2 introdujo el mensaje ***informRequest*** para permitirle a un manager comunicarle información de su MIB a otro manager y ***GetBulkRequest*** para recuperar de manera más eficiente gran cantidad de información desde un agente. Además de estas incorporaciones, se revisó el formato del PDU del trap de la versión SNMPv1 y se le dio un formato igual al del resto de los PDUs en la nueva versión.
La **Figura 3.13** muestra un único formato definidos para todos los tipos de PDUs del SNMPv2.

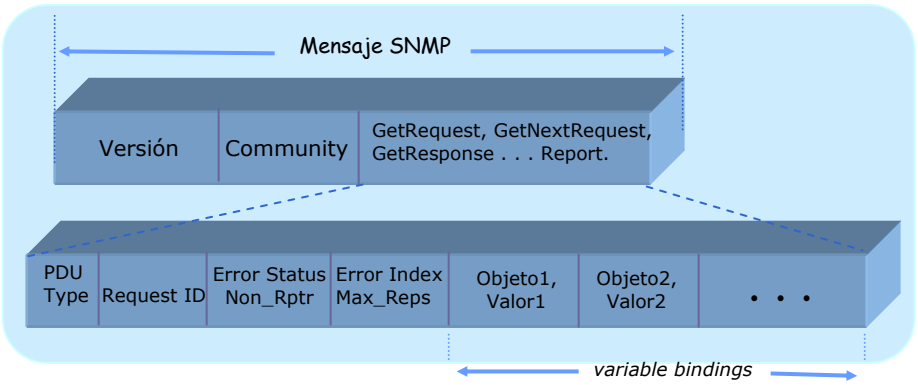



**Figura 3.13: Formato del PDU del SNMPv2.**

La estructura del PDU es muy similar a SNMPv1. Lo que cambió es el contenido de los campos: se agregaron nuevos códigos de error -*Error Status*- y se reutilizaron los campos de error de la versión anterior para transmitir dos valores *Max_Reps* y *Non_Rptr* [Case, J.; RFC1905]. Estos campos solo son utilizados por el nuevo mensaje ***GetBulkRequest*** para especificar la cantidad de repeticiones requeridas para recuperar las variables y cuántas de las variables no serán procesadas repetidas veces.
Como ya hemos analizado, el mensaje *GetNextRequest* permite recuperar el próximo objeto al OID suministrado en la *variable-bindings* del mensaje. Pero ¿qué pasa cuando el administrador quiere no sólo el objeto siguiente, sino también los que están después de ese? El ***GetBulkRequest*** resuelve esta situación recorriendo con un orden lexicográfico la MIB tantas veces como lo indique el parámetro *Max_Reps.*
Un ***GetBulkRequest*** ayuda a optimizar la recuperación de gran cantidad de información, lo que fue uno de los principales problemas del Snmpv1. En la primera versión del protocolo cuando una estación necesitaba recuperar mucha información, debía hacerlo usando reiteradas veces el mensaje GetNextRequest, por el contrario, con el

***GetBulkRequest*** alcanza con una solicitud para recuperar eficaz y rápidamente tablas de gran tamaño.
SNMPv2 también debió abordar problemas de seguridad del SNMPv1 y es en este aspecto en donde SNMPV2 se encontró con obstáculos significativos en la normalización. Por este motivo, a esta versión también se la nombra como SNMPv2C –c de *community*- porque en la práctica, la seguridad del protocolo siguió basada en *community strings*. La propia RFC 2570 [Case, J.; RFC2570] dice que: *"el framework SNMPv2, está incompleto ya que no cumple con los objetivos del diseño original del Proyecto SNMPv2. Los objetivo incumplidos están relacionados con aspectos de seguridad . . . ".* La versión 3 es la que logra terminar de incorporar seguridad al SNMP, por lo que analizaremos estos aspectos directamente –aunque no son exclusivamente del SNMPv3- en el próximo inciso.

## 3.5 Simple Network Management Protocol version 3 (SNMPv3)

SNMPv3 es la versión más nueva del SNMP. Esencialmente puede ser pensada como SNMPv2c más aspectos de seguridad y control de acceso. Esto significa que retiene las mismas operaciones de gerenciamiento del SNMPv2c pero introduce lineamientos para transportar mensajes SNMP con parámetros de seguridad que hacen del SNMPv3 un protocolo seguro y definen políticas de control de acceso a los datos de la MIB.
SNMPv3 incluye tres importantes servicios: autenticación, privacidad y control de acceso.

Las RFCs centrales del nuevo SNMP están detalladas en la tabla de la Figura 3.14.

| Número | Titulo | Fecha |
|---|---|---|
| **RFC 3411** | An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks | 2002 |
| **RFC 3412** | Message Processing and Dispatching for the the Simple Network Management Protocol (SNMP) | 2002 |
| **RFC 3413** | Simple Network Management Protocol (SNMP) Applications | 2002 |
| **RFC 3414** | User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3) | 2002 |
| **RFC 3415** | View-Based Access Control Model (VACM) for SNMP | 2002 |
| **RFC 3826** | The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model | 2004 |

**Figura 3.14: RFCs de SNMPv3**

Esta nueva versión del SNMP incorpora básicamente dos módulos para proveer seguridad y control de acceso. El primero de ellos conocido como UMS –del inglés User-Based Security Model - se encarga de la autenticación, cifrado y descifrado de paquetes SNMP y el segundo conocido como VACM -del inglés View-based Access Control Model- se encarga de administrar el acceso a los datos de las MIBs.

En la **Figura 3.15** ilustra la estructura de los mensajes SNMPv3. Analicemos por ahora los primeros campos del mensaje SNMPv3:

- ***msgVersion:*** la versión del paquete SNMP (1, 2 o 3).
- ***msgID:*** El ID del mensaje usado para coordinar los mensajes entre el egente y el gerente.
- ***msgMaxSize:*** El máximo tamaño de mensaje que el que el emisor es capaz de aceptar desde otra entidad SNMP.
- ***msgFlags:*** sirve para identificar si el mensaje ha sido procesado, autenticado y encriptados.
- ***msgSecurityModel:*** especifica el modelo de seguridad usado para generar el mensaje. De esta manera la entidad que recibe el mensaje sabe qué modelo de seguridad usar para procesar la seguridad del mensaje.

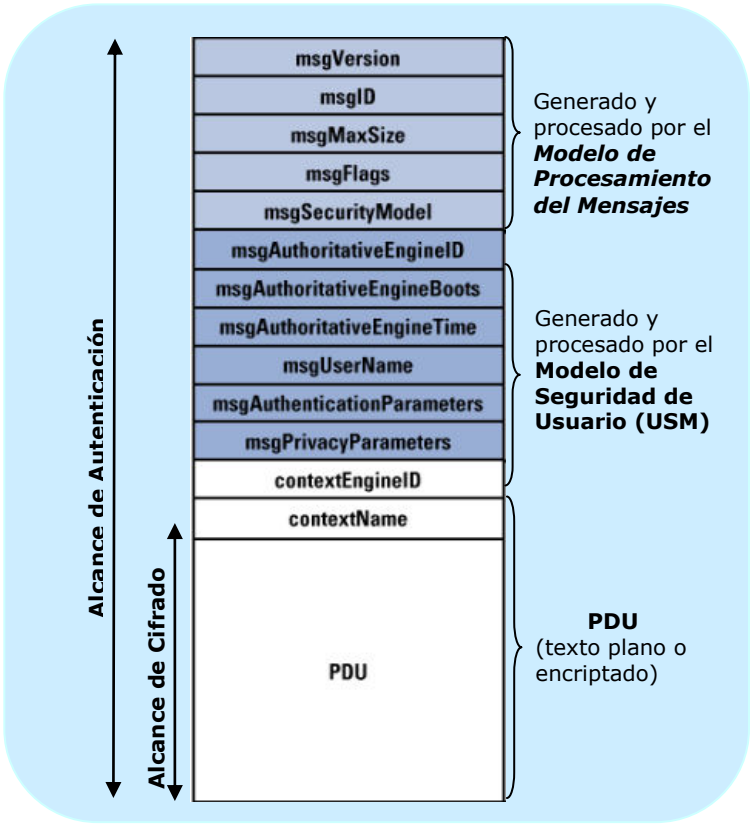



**Figura 3.15: Formato del mensaje SNMPv3 con el Modelo de Seguridad basado en Usuario**

Para incorporar aspectos de seguridad fue necesario, como podemos apreciar en la Figura 3.15, incorporar nuevos campos al paquete. Analizaremos el uso de los campos a medida que se introducen los nuevos modelos.

### 3.5.1 Modelo de seguridad basado en usuario –USM-

Un punto débil de todas las anteriores versiones de SNMP ha sido la falta de un sólido y consensuado esquema de seguridad. En el diseño del modelo de seguridad basada en usuario o USM [Stalling W.], se ha trabajado para brindar:

- ***Integridad de Datos*:** Asegurar que los datos no sean maliciosamente alterados durante el tránsito por una entidad no autorizada.
- ***Autenticación del origen de datos*:** Conocer exactamente de quién y de dónde provienen los datos para evitar que una entidad no autorizada asuma la identidad de un usuario autorizado.
- ***Confidencialidad de los datos***: Asegurar que una entidad no autorizada no pueda observar los intercambios de datos.

El USM es capaz de proteger los paquetes SNMPv3 de las amenazas descriptas mediante la utilización de un concepto de múltiples usuarios, donde cada usuario proporciona las ***claves secretas*** para la autenticación y privacidad. Los protocolos de *autenticación* especificados para su uso son *HMAC-MD5* y *HMAC-SHA*. El protocolo de *privacidad* especificado es *CBC-DES*. La RFC establece que los protocolos de seguridad utilizados para la USM se consideran aceptablemente seguros.

El framework SNMPv3 introduce nuevos conceptos que analizaremos antes de abordar el formato de los paquetes SNMPv3. Algunos de los conceptos útiles para USM son:

- ***snmpEngineID:*** un identificador único para un motor o entidad SNMP.
- ***snmpEngineBoots:*** el número de veces que la entidad SNMP se ha reiniciado desde que el snmpEngineID ha sido seteado.
- ***snmpEngineTime:*** El número de segundos desde que el contador snmpEngineBoots fue incrementado.
- ***snmpSecurityLevel:*** Existetn tres posible niveles de seguridad: noAuthNoPriv –sin autenticación ni privacidad-, authNoPriv –con autenticación sin privacidad- y authPriv- serán ejecutados controles de autenticación y de privacidad-.
- ***Authoritative SNMP Engine:*** A fin de proteger contra la repetición de mensajes, el retraso, y la redirección, uno de los motores de la comunicación SNMP es designado como el motor de SNMP autorizado. El motor SNMP autorizado es aquel que recibe mensajes de SNMP, que necesita una respuesta. Este es siempre el agente que se comunica con un gestor.

Ahora que analizamos estos nuevos conceptos, podemos continuar con el formato de los mensajes del la **Figura 3.15**. Los parámetros utilizados por Modelo de Seguridad de Usuario son:

- ***msgAuthoritativeEngineID:*** es un ID que identifica cuál es el *engine SNMP* -la entidad SNMP- autorizada, no importa de qué lado se origina el paquete.
- ***msgAuthoritativeEngineBoots y msgAuthoritativeEngineTime:*** son enteros que representan el número de veces que el motor SNMP se reinició y la cantidad de segundos desde la ultima reinicialización.
- ***msgAuthoritativeEngineTime:*** The snmpEngineTime of the authoritative engine. El objeto snmpEngineTime es un número entero que representa el número de segundos

desde esta autorizada motor SNMP último incrementa el objeto snmpEngineBoots. Cada motor SNMP autorizado se encarga de incrementar su valor propio snmpEngineTime una vez por segundo.

- ***msgUserName:*** El nombre del usuario cuyas claves secretas fueron usadas para autenticar y encriptar el paquete.
- ***msgAuthenticationParameters:*** Si el paquete ha sido autenticado, este campo contiene el HMAC-MD5 o el HMAC-SHA computado para el paquete.
- **msgPrivacyParameters:** Si el *scopedPDU* del paquete ha sido encriptado, entonces este campo contiene el ***salt*[2]*,*** dato que fue usado como entrada en el algoritmo de DES.

***Autenticación***

El USM especifica el uso de los algoritmos *MD5 -Message Digest 5–* y del *SHA-1 -Secure Hash Algorithm 1-*. Estos algoritmos son usados para crear los *digest* únicos para cada mensaje. Ninguno de estos algoritmos puede ser usado directamente como un código de autenticación del mensaje porque no usan claves secretas como entrada para derivar el digest del mensaje. Por esto, se usa el algoritmo *HMAC -Keyed Hashing for Message Authentication-* y *SHA-1 o MD5* para computar los digest del mensaje. El algoritmo HMAC define un procedimiento para agregar una clave secreta a los datos y entonces computa el MD5 o SHA-1. Esto garantiza que las partes que deseen computar idénticos digest para el mismo block de datos, deban compartir una clave secreta.

***Privacidad***

El USM de SNMPv3 permite a los managers y agentes *encriptar* mensajes para prevenir ataques de seguridad por partes de terceras partes. Para lograrlo, el manager y el agente deben compartir una clave secreta. Cuando se acuerda usar privacidad todo el tráfico entre el manager y el agente es encriptado usando el algoritmo DES. La entidad que envía encripta el mensaje entero usando DES y la clave secreta y el que recibe desencripta usando DES y la misma clave secreta. USM especifica el uso del algoritmo CBC-DES *-Data Encryption Standard* (DES) en el modo de operación *Cipher Block Chaining* (CBC)- para cifrar y descifrar los paquetes SNMPv3. El ámbito de aplicación de esta codificación sólo cubre el *scopedPDU* que contiene los datos del PDU y el contexto utilizado por el VACM. Las claves secretas se almacenan en la tabla de usuarios y el ***salt*** se transmite con el paquete en el campo ***msgPrivacyParameters***.
Si bien la RFC 3414 describe el uso HMAC-MD5 y HMAC-SHA como los algoritmos iniciales de autenticación, y el uso de la CBC-DES como el algoritmo inicial para privacidad, el USM permite el uso de otros algoritmos. La RFC 3826 describe el uso del *AES - Advanced Encryption Standard-* como un algoritmo de privacidad alternativo al *DES*.

### 3.5.2 Modelo de control de acceso basado en vistas o VACM –del inglés View-based Access Control Model-

[2] Salt: es una de las entradas usadas por el algoritmo DES además del la clave secreta y los datos a encriptar.

Ahora, si la seguridad de SNMPv3 es la descripta arriba, tenemos un problema. Si se quieren asignar diferentes niveles de privilegio a usuarios: por ejemplo si queremos permitir *resetear/rebootear* un dispositivo remoto a un administrador y prevenir de esto a los usuarios ordinarios a la vez que queremos que todos puedan leer el estado del agente, no es suficiente con categorizar a los usuarios dándoles a ellos autenticación y privacidad de claves.
Para esto es útil VACM, el cual usa una MIB para especificar por usuario a que parte de la MIB del agente puede acceder bajo determinadas condiciones. Las condiciones incluyen, ***nivel de seguridad*** -por ejemplo: una parte de la MIB podría ser accedida con request autenticado y otra parte con request autenticado y encriptado-, ***modelo de seguridad*** -por ejemplo: no permitir que managers SNMPv1 y SNMPv2 accedan a las tablas de configuración de SNMPv3-, ***userName*** -por ejemplo: "markus" podría acceder a todos los objetos de la MIB y "pupi" sólo a dos subárboles-, ***viewTypes*** -por ejemplo: un usuario podría tener privilegios para leer pero no para escribir ciertos objetos- y en cuál contexto existe el objeto.
VACM no especifica accesos para instancias simples de objetos de la MIB sino para subárboles de la MIB del agente. Un subárbol es un conjunto de todos los objetos que tienen un prefijo común en los identificadores de objetos.
El subsistema de control de acceso de una entidad SNMP tiene la responsabilidad de chequear si está permitido el acceso a un objeto por parte de otra entidad. El VACM se elaboró sobre el concepto de *Community String* pero con un modelo de control de acceso más estricto y más dinámico.
Volviendo al formato del paquete ilustrado en la **Figura 3.15**, el VACM usa los campos ***msgFlags***, ***msgSecurityMode*** y ***scopedPDU*** para determinar el acceso al mensaje. El campo ***msgFlags*** es usado para determinar el nivel de seguridad del mensaje -noAuthNoPriv, authNoPriv, or authPriv-. El ***msgSecurityModel*** se utiliza para especificar el modelo de seguridad usado para asegurar el mensaje. El ***scopedPDU*** contiene el contexto y el PDU con los datos del mensaje y está subdividido en tres partes:

- ***contextEngineID:*** es un ID que identifica unívocamente a una entidad SNMP que puede acceder a una instancia de un objeto gerenciado dentro de un contexto.
- ***contextName:*** es el nombre único de contexto al que pertenece administrativa un PDU.
- ***PDU (data):*** Es la unidad de datos del protocolo que contiene una operación y la *variable-bindings*. El acceso a cada variable es chequeado en forma individual.

El manual *Hands-On SNMPv3 Tutorial & Demo*, de NuDesign Technologies, Inc., disponible en http://www.ndt-inc.com/SNMP/pdf/NuDesign_SNMPv3_Tutorial_&_Demo _Manual.pdf contiene información mas detallada sobre los aspectos de seguridad incorporados en la versión SNMPv3.

## Síntesis

El SNMP es quizá el protocolo de gerenciamiento más conocido. Es además, el protocolo de gerenciamiento que más se ha adoptado para aplicaciones de monitoreo. SNMP se basa en la idea de que la gestión de la información se organiza en MIBs, con variables de gerenciamiento individuales -u objetos gerenciados-, identificados a través de

identificadores de objetos (OID). SNMP proporciona un pequeño conjunto de primitivas que permiten a un administrador leer y escribir valores en un MIB, y un agente para enviar eventos.
SNMP tienen tres versiones, cada una superadora de su antecesora, sin embargo, hoy en día se puede encontrar dispositivos usando cualquiera de ellas. El SNMP original, ahora conocido como SNMPv1, es la versión más simple y, para los agentes, la más fácil de implementar. SNMPv2c añade nuevas funcionalidades, siendo la más importante la incorporación de un medio eficaz para recuperar gran cantidad de información de gestión. Luego aparecieron otras versiones intermedias como SNMPv2u y SNMPv2* con algunas mejoras pero ninguna fue ampliamente aceptadas por la carencia de consenso y porque se percibían deficientes definiciones.
Por último, aparece SNMPv3 que termina de resolver la falta de seguridad -que comienza a direccionarse los SNMPv2- convirtiéndose en un protocolo mucho más completo, pero también, más complejo.

# Capítulo 4

## Análisis de librerías relacionadas con SNMP

### 4.1 Introducción

En este capítulo se describen un conjunto de herramientas relacionadas con SNMP, se analizan ventajas y desventajas de cada una de ellas con el propósito de seleccionar una de ellas para la implementación del prototipo. Todas las librerías implementan o brindan facilidades para comunicarse con agentes SNMP. En primera instancia se sintetiza CMU SNMP una implementación de SNMP, luego se describen más detalladamente Java Dynamic Management Kit (JDMK), una arquitectura para diseñar sistemas distribuidos, que permite incorporar inteligencia en los agentes y provee una abstracción de la capa de comunicación. En esta API, conocida como JMX, fue finalmente incorporada con mejoras a las nuevas versiones de la plataforma JAVA.

### 4.2. Herramientas de desarrollo evaluadas para gerenciamiento de Redes

Las herramientas evaluadas para la propuesta de una arquitectura e implementación de un prototipo fueron las siguientes: CMU_SNMP, Adventnet, JMAPI, JDMK y JMX. Describiré brevemente las características, ventajas y desventajas de cada una de ellas, indicando cuál seleccioné para esta tesis y justificando el por qué de su elección.

#### 4.2.1 CMU_SNMP

CMU-SNMP es una herramienta que provee una implementación del SNMP. Cuenta con un conjunto de programas simples y una biblioteca de funciones que permiten la implementación de sistemas basados en SNMP más complejos [CMU-SNMP]. Su nombre, CMU deriva de la Universidad en donde fue desarrollada, Carnegie Mellon University.
Los programas que incluye son simples e incompletos: snmpget, snmpgetnetx, snmpnetstat, snmpstatus, snmptest, snmptrap, snmptrapd y snmpwalk, están escritos en C y se deben ejecutar desde la línea de comando. A continuación se detalla el uso de algunos de estos programas:

***snmpget/snmpgetnetx***

Esta aplicación utiliza el ***GetRequest*** del SNMP para consultar información de una entidad de red. Su sintaxis es:

```
snmpget host community variable-name [ variable-name] ...
```

Donde:
- **host:** representa la entidad a ser consultada. Puede ser informada tanto por el nombre como por la dirección IP;
- **community:** especifica el nombre de la comunidad para la transacción del sistema remoto;
- **varible-name:** representa los identificadores del objeto.

***snmpnetstat***

Este programa recupera una lista de valores desde un sistema remoto utilizando el protocolo SNMP. Tiene varios tipos de salidas de acuerdo a lo especificado en el requerimiento:

```
snmpnetstat host community [-an]
snmpnetstat host community [-inrs]
snmpnetstat host community [-n] [-I interface] interval
snmpnetstat host community [-p protocol]
```

Donde:
- **host:** representa la entidad a ser consultada. La entidad puede ser identificada tanto por el nombre como por la dirección IP;
- **community:** especifica el nombre de la comunidad para la transacción del sistema remoto;

De las 4 formas detalladas, la primera muestra una lista de *sockets* activos. La segunda los valores relativos a opciones especificadas. La tercera muestra el tráfico de paquetes en una interface y la última muestra estadísticas sobre un protocolo determinado.

***snmptest***

Esta es una aplicación que permite monitorear una entidad SNMP de una red.

```
snmptest host community
```

Donde:
- **host:** representa la entidad a ser consultada. Puede ser identificada tanto por el nombre como por la dirección IP;
- **community:** especifica el nombre de la comunidad para la transacción del sistema remoto;

Al invocar este programa, el intérprete en línea de comando hace ciertos requerimientos al usuario, como la variable a ser monitoreada. La primera vez que se invoca se asume

que se hace un requerimiento de tipo GET. Esto puede ser cambiado tipeando "$N" para que el próximo requerimiento sea de tipo GETNEXT y "$S" para ejecutar un SET.
Como puede observarse la interfaz de usuario es muy pobre, completamente textual y requiere de un entrenamiento por parte del usuario para su uso.

### 4.2.2 Adventnet

Adventnet Web NMS [WebNMS] es una solución basado en Java para el gerenciamiento de redes. Provee un conjunto integrado de herramientas que pueden ser utilizadas para implementar aplicaciones avanzadas para administrar redes de datos. Provee un *framework* que integra SNMP, la Web y tecnologías Java.
Este ambiente provee importantes funciones de gerenciamiento tales como:

- Descubrimiento de la topología de la red.
- Mapas de red funcionales basados en browser.
- Manejo de fallas, seguimiento de eventos.
- Reportes de performance y recolección de datos.
- Gerenciamiento y monitoreo de dispositivos.
- Browser de las Mibs.

Su arquitectura responde al modelo de 3 capas, con una Base de Datos, un Servidor para el gerenciamiento y clientes para facilitar la interacción del usuario (administrador de la red).
Para la Base de Datos, *Advent-Net* utiliza un modelo orientado a objetos muy sencillo para la representación de los objetos almacenados en la base de datos. Las aplicaciones pueden extender este modelo y adicionar objetos y atributos de objetos dinámicamente para soportar necesidades particulares.
La Base de datos tiene los siguientes objetos:

- El objeto **`Topo`**, es la clase base para todos lo objetos de la base
- **`Network`**, es la clase que representa la red.
- **`Node`**, clase que representa a un nodo de la red.
- **`IpAddress`**, que representa a una dirección IP.

El servidor consiste en un conjunto de componentes basados en Java. Básicamente, es un servidor web que permite y controla el acceso remoto de navegadores estándares. El Servidor incluye componentes para el descubrimiento de la topología de la red, servidores de eventos y mapas. Además, provee una comunicación con los agentes gerenciados vía SNMP, Corba, http y RMI. Las componentes de Adventnet compatibilizan adecuadamente con las componentes de JavaBeans.
El cliente es un programa que debe iniciarse desde un navegador web. Todas las funciones de gerenciamiento y administración pueden ser accedidas desde lugares remotos a través de un navegador.

Al momento de evaluar Adventnet se encontraron las siguientes limitaciones:

- No proveía configuración de parámetros para los servlets de mapas, alertas, eventos, browsers de Mibs y reportes.
- Era inestable, se conocían múltiples problemas (*bugs*) no resueltos.

- No es producto *open source*. De algunas componentes se provee algunas licencias por 45 días.

A pesar de estas limitaciones se evidencia, a diferencia del anterior, un producto con constante crecimiento.

### 4.2.3 Java Management Application Programming Interface –JMAPI-

JMAPI fue el primer intento realizado por SUN para crear un framework que se comporte como las infraestructuras de gerenciamiento tradicionales en ese momento, tales como, Tivoli (IBM) u Open View (Hewlett_Packard). Este intento apuntaba al desarrollo de una herramienta implementada completamente en Java a diferencia de las soluciones propietarias existentes en ese momento.
Un grupo de administradores e ingenieros de SunSoft, una división de SUN conocidos como RMTC (Rocky Mountain Technology Center) trabajaron en este proyecto. El RMTC que funcionaba en Colorado Spring, anunciaron el lanzamiento de JMAPI en el año 1995. El equipo de trabajo sólo contaba, para la creación de JMAPI, con las tempranas versiones del AWT[1]. Este no brindaba las componentes de GUI necesarias para una buena consola de gerenciamiento, por tal motivo los ingenieros del RMTC construyeron una gran librería de clases llamada *Admin View Module (AVM)*, la cual extendía el AWT, para potenciar sus características. Además, incorporaron un subsistema de Ayuda (JavaHelp) que finalmente no benefició el desarrollo de sistemas para gerenciamiento.
Todo este gran proyecto pronto quedó opacado por la propia labor de SUN, debido a las sucesivas versiones del Java Developers Kit (JDK), principalmente por la incorporación de las componentes Swings[2] al JDK, que superaban ampliamente a las clases desarrolladas por el RMTC. Como era de esperar, a fines de 1996, SUN anunció el cierre del RMTC y posteriormente, la transferencia del proyecto a JavaSoft de Francia.
A pesar de todos estos problemas JMAPI logró alcanzar su objetivo principal, desarrollar un *framework* completamente escrito en Java.
JMAPI provee un conjunto herramientas y componentes que permiten desarrollar aplicaciones/applets para gerenciamiento de redes de datos.
Este API provee componentes para la construcción de interfaces de usuarios más gráficas y de mejor calidad, como:

- Botones con imágenes
- Barras de herramientas
- Visualización de jerarquías
- Gráficos de barras, tortas, etc.
- Tablas
- Ventanas de diálogo

[1] AWT -Abstract Window Toolkit-: fue incluido en el JDK 1.0, y provee a los programadores con una librería rudimentaria para construir applets y aplicaciones con interfaces de usuario muy simples. Incluye componentes de interacción simples como botones, listas y texto, con un mínimo conjunto de *fonts* y carentes de imagen.
[2] Swing extiende el original AWT agregando un conjunto de clases para construir interfaces de usuario gráficas de alta claidad. Fueron incorporados como parte de la plataforma java en el JDK 1.2.

JMAPI consiste básicamente de 3 componentes funcionales: un BUI -Browser User Interface-, una ARM -Admin Runtime Modulo- y algunas aplicaciones.
La BUI es el medio por el cual los administradores pueden realizar las consultas. Las *applets* JMAPI consisten del módulo AVM, el cual provee componentes para la construcción de la GUI.
El ARM es el módulo que permite concentrar la administración de la red y está compuesto por un Server HTTP, una fábrica de objetos gerenciados que implementan las operaciones de gerenciamiento y administración. Esta fábrica accede a una Base de Datos relacional que actúa como repositorio de la información de gerenciamiento.

### 4.2.4 Java Dynamic Management™ Kit -JDMK-

Java Dynamic Management Kit [JDMK] es una interface de programación (API) con algunas herramientas que facilitan el desarrollo de aplicaciones de gerenciamiento. JDMK implementa la especificación de las APIs: JMX (Java Management Extensions) y JMX Remota, brindando un *framework* para gerenciamiento de objetos java, a través de aplicaciones basadas en tecnología java.
Además, Java DMK provee una API para SNMP, que facilita el desarrollo de agentes y gerentes que puedan comunicarse con sistemas de gerenciamiento existentes. La última versión disponible es JDMK 5.0 que incluye soporte para SNMPv3, basada en las RFCs 2571, 2572, 2574, y 2576. Como una implementación de la especificación JMX, el producto provee una *framework* para gerenciar objetos java, a través de aplicaciones basadas íntegramente en tecnologías Java.
Java DMK provee una arquitectura para diseñar sistemas distribuidos, que permite incorporar inteligencia en los agentes y provee una abstracción de la capa de comunicación. Para la aplicación de gerenciamiento –y también para el agente- se pueden usar otras APIs tales como las componentes Swing para crear interfaces de usuario gráficas o JDBC para acceder a Bases de datos. JDMK no provee componentes para interfaz de usuario para la aplicación de gerenciamiento y no maneja persistencia de datos. Java DMK incluye:

- Una arquitectura de gerenciamiento: conforme a la especificación de la API JMX para el nivel de agente y de la instrumentación del agente.

- Módulos de comunicación: jdmk define una API para acceder a agentes JMX remotamente. El producto incluye módulos para comunicación basados en RMI, HTTP y HTTPS. También provee un *HTML Adaptor*, el cual soporta acceso a un agente desde un navegador. Este protocolo permite ver el estado del mismo, la información mostrada es estática y no persistente.

- Servicios para el agente: incluye APIs para facilitar el monitoreo, descubrimiento de agentes y componentes para implementar mecanismos de seguridad (en la versión 5.0).

- Una API para SNMP: permite desarrollar agentes/gerentes basadas en SNMP que puedan comunicarse con sistemas de gerenciamiento existentes, ayudando a los mismos, a migrar hacia una arquitectura más dinámica.

La API para SNMP es parte de JMX y provee un conjunto de clases java que ayudan a desarrollar aplicaciones para manejar gerentes SNMP, agentes SNMP y *proxies* SNMP.
Esta API está dividida en dos paquetes:
El paquete **`javax.management.snmp.manager`** contiene las clases e interfaces que constituyen las principales componentes de la API. Estos son los objetos que deben ser implementados o instanciados para manejar agentes, parámetros, sesiones, listas de variables (varBind) y traps.
El paquete **`javax.management.snmp`** contiene las clases e interfaces que complementan el paquete anterior. Estas clases permiten manejar variables individuales, strings, paquetes PDU, etc.

Las principales clases o componentes que conforman estos paquetes son SNMPPeer, SNMPParameters, SNMPSession y SNMPRequest. A continuación se detallan las características de cada una de ellas:

**Agentes/Dispositivos SNMP (SNMPPeers)**

Un SNMP Peer es usado para representar a los agentes remotos. Cada agente remoto es representado por un simple objeto *SnmpPeer*. Este objeto puede crearse con dirección IP, nombre de host y port del agente remoto, o simplemente con su dirección IP o su nombre. Un objeto SnmpPeer podría contener cierta información como dirección IP del agente remoto, #port del agente remoto, cantidad máxima de OIDs que pueden ser codificados en un requerimiento SNMP, tamaño máximo de un paquete, etc.

**Sesiones SNMP (SNMPSession)**

Esta clase permite controlar una sesión entre managers y agentes (peers). Un objeto SNMPSession crea y maneja requerimientos SNMP para múltiples peers. Esta clase provee métodos para instanciar los diferentes tipos de mensajes del SNMP, de una forma sincrónica o asincrónica.
Cuando se envía un requerimiento sincrónico, la sesión puede enviar sólo uno a la vez. El manager debe esperar la respuesta desde el agente antes de producir un nuevo requerimiento. Si bien este modo es más fácil de implementar, tiene una taza limitada de requerimientos. El tiempo de espera por una respuesta puede ser programado con Java para poder optimizar los requerimientos sincrónicos. Por el contrario, cuando se envía un requerimiento asincrónico, la sesión no necesita esperar por la respuesta para enviar otro requerimiento. En este caso, la sesión mantiene una lista de todos los requerimientos y respuestas. Esto le permite al manager manejar un número mayor de requerimientos y agentes simultáneamente en la misma sesión. Los requerimientos son reenviados después de un determinado tiempo si no son respondidos.

**Requerimientos SNMP (SNMPRequest)**

Esta clase permite manejar todos los requerimientos enviados en una sesión. Un objeto de esta clase se crea cuando se efectúa alguno de estos requerimientos SNMP: *snmpGet, snmpGetNext, snmpSet, snmpGetBulk, snmpGetPoll, snmpGetNextPoll y snmpWalkUntil*.

**Parámetros SNMP (SNMP parameters)**

Esta clase contiene información sobre las *communities* para lectura y escritura de SNMP y la versión de SNMP. Esta información es usada por los objetos SnmpSession mientras intercambia paquetes con un SnmpPeer. Un objeto de esta clase puede ser compartido por múltiples objetos SNMPPeer. Por defecto, un manager no tiene acceso de escritura sobre un objeto y pertence a la comunidad pública para lectura, sin embargo estos accesos pueden ser configurados de cualquier manera.

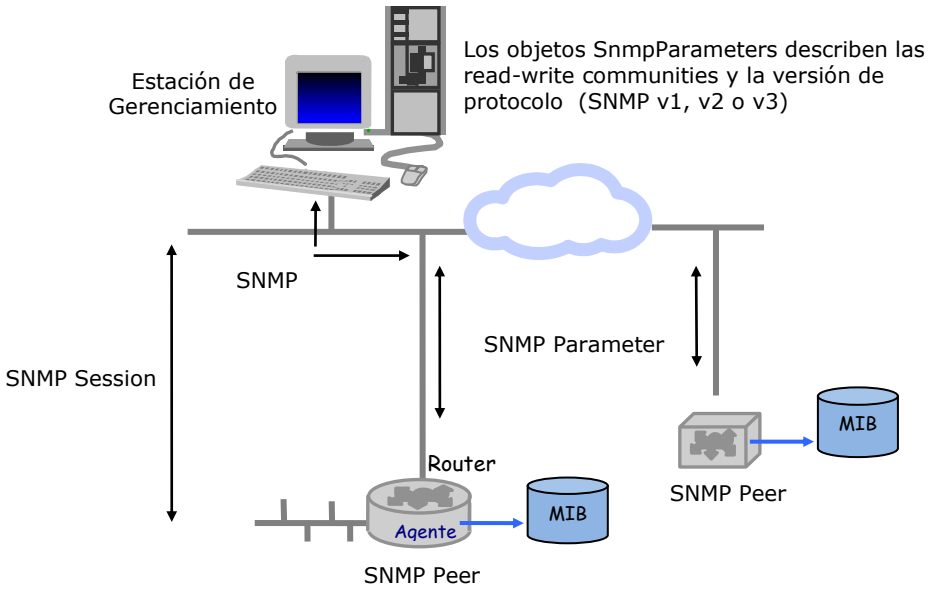



**Figura 4.2: Clases de la API para gerenciamiento de SNMP**

El Java DMK fue la primera implementación comercial del estándar JMX para construir soluciones para gerenciamiento basadas en JAVA. Parte de esta implementación de SUN fue incluida en la plataforma estándar de JAVA a partir de versión 5.
A partir de la versión J2SE 5.0, la plataforma incorpora soporte para gerenciamiento y monitoreo para la plataforma java. Esta API básicamente provee:

- *Instrumentación para la Java Virtual Machine (JVM):* la plataforma JAVA incluye un agente que permite monitorear la JVM: sus *threads*, uso de memoria heap, etc.
- *Herramientas para monitorear agentes:* la plataforma provee una herramienta para monitoreo llamada *JConsole*. Esta usa el nivel de instrumentación JMX para modelar la JVM y para proveer información sobre *performance* y el consumo de recursos de aplicaciones corriendo sobre la plataforma java.
- Una *API para monitoreo y gerenciamiento:* la API implementa la especificación de JMX. JDMK, es un producto de SUN y mucha de la experiencia ganada con ese producto y de las implementaciones han sido volcadas a la plataforma JAVA. Esta API, conocida como *Java Management Extensions o JMX* está formada por un conjunto de paquetes que definen la arquitectura, los patrones de diseño, las interfaces y los servicios para

monitoreo y gerenciamiento de aplicaciones y redes con java. Están basados en la especificación JMX [JSP3] [JSP160]. En el próximo capítulo se describe en detalle esta API.

### 4.3 Cuadro comparativo de APIs evaluadas

El siguiente cuadro resume las características más destacables de cada API:

| Herramienta | Multiplataforma | IDE | Lenguaje | Desventajas |
|---|---|---|---|---|
| **CMU_Snmp** | No | No | C++ | No es multiplataforma.<br>Funcionalidades limitadas.<br>Comunicación únicamente con CGI |
| **Adventnet** | No | Si | Java | Software disponible solo para el Sistema Operativo Windows.<br>No provee configuración de parámetros para alertas y eventos.<br>No provee aspectos de seguridad como SSL.<br>Licencia: Freeware. No es gratuita. |
| **JMAPI** | Si | No | Java (1.0) | Su instalación es compleja y debe ser sobre NT Server exclusivamente.<br>No es gratuita. |
| **JDMK** | Si | No | Java 2 | Es un producto de SUN. No es gratuita.<br>La interfaz de usuario son páginas HTML.<br>No existe información persistente => no se pueden hacer análisis de *performance*, estadísticas, etc.<br>**Ventajas respecto de los anteriores:**<br>- Permite manejar seguridad (SNMP v3)<br>- Permite agregar nuevas MIB sin detener el agente.<br>- Facilita la implementación de agentes java para monitoreo. |
| **JMX en J2SE 5.0** | Si | Net Beans | Java 2 | Tiene las mismas ventajas de JDMK.<br>Es libre, es una librería dentro de la API de java, a partir de la versión J2SE 5.0.<br>Dispone de una Consola para gerenciamiento –JConsole- para monitoreo de objetos JAVA. |

**Figura 4.1: Cuadro comparativo de herramientas analizadas**

En el cuadro se sintetizan las ventajas y desventajas de cada una de las soluciones analizadas. En primera instancia se evaluó CMU-SNMP, una implementación de SNMP que además de ser muy limitada, fue descontinuada.
Luego se analizó Adventnet, un producto comercial de una empresa pionera en aplicaciones para gestión de redes de datos. Esta empresa también ha sido promotora de los estándares para gerenciamiento de aplicaciones Java, se ha unido al grupo de

expertos del Java Community Process[3] para participar de la definición de Java Management Extensions (JMX), el estándar clave para la gestión de aplicaciones Java y J2EE. En los últimos años, esta empresa ha implementado nuevas soluciones basadas en JMX después de reconocer el potencial de esta API como una norma para gestión basada en estándares para redes y aplicaciones Java.

Posteriormente se analizó JMAPI, el primer intento de parte de SUN por conseguir un framework para gerenciamiento escrito en JAVA a diferencia de las soluciones propietarias existentes en ese momento. Este proyecto tuvo muchos problemas pero logró alcanzar su objetivo principal, desarrollar un *framework* completamente escrito en Java.

Finalmente se comenzaron a desarrollar las primeras pruebas con las versiones trial de JDMK hasta que finalmente, después de la aparición del JSE 5.0 se utilizó la plataforma estándar únicamente.

Después de analizar estas herramientas decido implementar el prototipo usando la API estándar de la plataforma JAVA denominada JMX.

## Síntesis

En este capítulo se han sintetizado las APIs analizadas para la implementación del prototipo. Después de haber realizado las primeras pruebas con ***cmu-snmp*** y ***Adventnet***, se comenzaron a probar las librerías de ***JDMK***. Por las características de estas tres APIs decidí ahondar en JDMK a pesar de no ser un producto de libre acceso, del cual debía obtener una versión de prueba nueva cada 30 días para poder trabajar. Finalmente cuando la versión J2SE 5.0 incorporó la API para gerenciamiento ***JMX*** –basada en JDMK- decidí realizar todas las pruebas para el prototipo con la plataforma estándar de JAVA.

[3] Java Community Process (JCP): es una organización que guía el desarrollo de las tecnologías JAVA así como también controla las especificaciones técnicas de la misma.

# Capítulo 5

## Arquitectura de Cafeto

### 5.1 Introducción

En este capítulo describo la arquitectura de Cafeto, la cual está basada en Java y destinada a proveer un escenario capaz de soportar heterogeneidad de recursos y monitoreo de datos altamente dinámicos, usando SNMP. La arquitectura que se presenta consta de un manager compatible con Java Management Extensions (JMX) [JSR 255, JMX] que se comunica con agentes JMX, los cuales funcionan como mediadores entre el manager y un conjunto de agentes SNMP.
La tecnología JMX fue desarrollada a través del Java Community Process (JCP)[1] como 2 especificaciones relacionadas (Java Specification Requests-JSRs):

- **JSR 3:** Java Management Extensions Instrumentation and Agent Specification.
- **JSR 160:** Java Management Extensions Remote API (Application Programming Interface).

La última especificación JMX es la **JSR 255** - **Java Management Extensions (JMX) Specification, version 2.0**[2]. Esta especificación actualiza la API anterior incorporando tipos genéricos y anotaciones para proveer un modelo de programación más simple.
Las primeras implementaciones de estas especificaciones se incluyeron por primera vez en la version 5.0 de la plataforma estándar de java; en la actualidad, la plataforma estándar **J2SE versión 7.0** incluye una implementación completa de la **JMX 1.4**.
A continuación haré un análisis de las componentes y servicios provistos por esta última especificación JMX y posteriormente presento una arquitectura basada en JMX para gerenciamiento de redes usando SNMP.

### 5.2 La especificación de Java Management Extensions (JMX)

La especificación JMX define una arquitectura para gerenciamiento y un conjunto de APIs que describen las componentes de esta arquitectura. Tal especificación define

1 Java Community Process (JCP): es una organización que guía el desarrollo de las tecnologías JAVA así como también controla las especificaciones técnicas de la misma.
2 Esta JSR actualiza las APIs JMX y JMX Remote para la versión 6.0 de la plataforma estándar (J2SE). Mejora las características existentes y agrega nuevas funcionalidades.

una arquitectura de tres niveles, que cubre funcionalidades para implementar agentes que puedan comunicarse con gerentes. Los niveles de esta arquitectura son:

- Nivel de instrumentación
- Nivel del agente
- Nivel de servicios distribuidos

Adicionalmente provee un conjunto de APIs para protocolos de gerenciamiento existentes. Estas APIs son independientes de los tres niveles, pero son esenciales para la comunicación con sistemas de gerenciamiento actuales.

La **Figura 5.1** muestra los niveles de la arquitectura JMX, donde pueden observarse las diferentes componentes de cada nivel y los protocolos de comunicación.

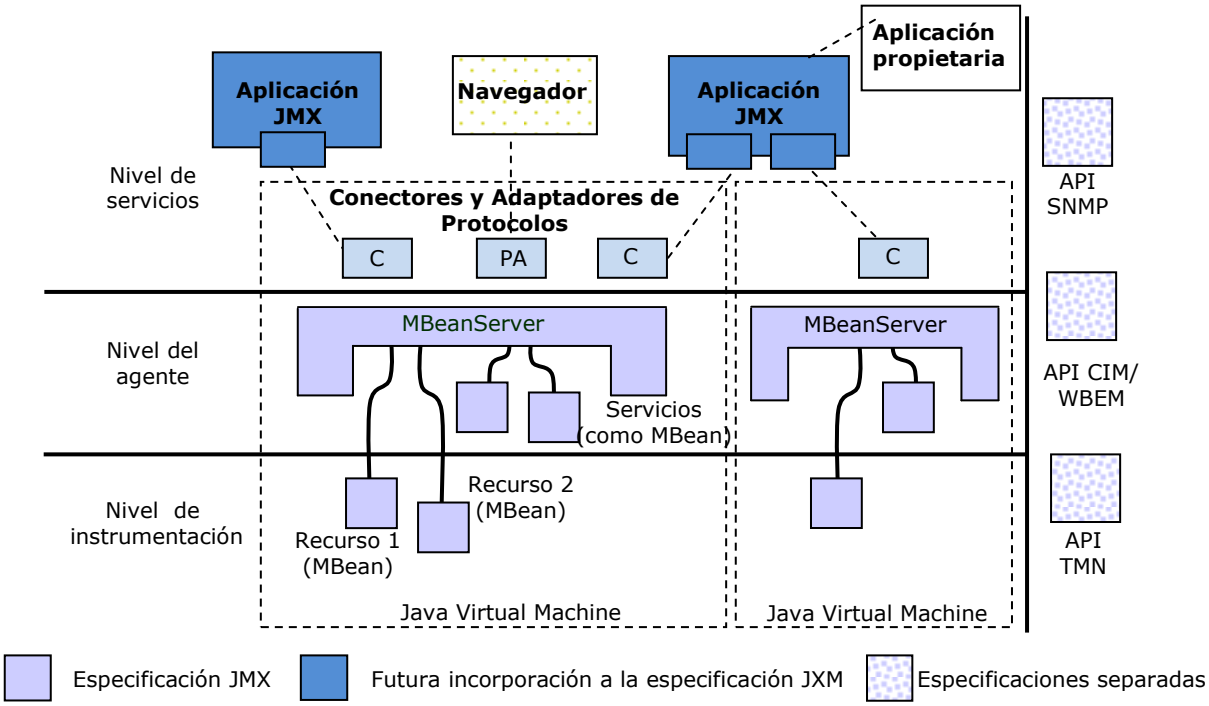



**Figura 5.1: Relación entre las componentes principales de JMX (imagen tomada de la especificación JMX y adaptada)**

### 5.2.1 Nivel de instrumentación (recursos gerenciados)

El nivel de instrumentación provee una especificación para implementar **recursos JMX gerenciables**. Un recurso JMX gerenciable puede ser una aplicación, una implementación de un servicio, un dispositivo, un usuario, etc. Son desarrollados en java y pueden ser manejados por aplicaciones JMX.
La instrumentación de un determinado recurso es provisto por uno o más *beans* gerenciados, conocidos como **MBeans**. Un MBean es un objeto Java gestionado, similar a un componente JavaBeans[3] [JavaBeans], que sigue los patrones de diseño establecidos en la especificación JMX. Los MBeans exponen una interfaz de gestión

[3] Una clase JavaBean es una clase debe obedecer ciertas convenciones sobre nomenclatura de métodos, construcción y comportamiento. Las convenciones requeridas son: tener un constructor sin argumentos, permitir la accesibilidad de sus propiedades a través de métodos get y set que siguen una convención de nomenclatura estándar e implementar la interface serializable.

que consiste en un conjunto de atributos y un conjunto de operaciones que les permite auto-describirse.
La instrumentación de un recurso, permite que el mismo sea gerenciado a través del nivel del agente, sin embargo, los MBeans no requieren información acerca del agente JMX con los que operará. Adicionalmente, este nivel especifica un mecanismo de notificación para propagar eventos, que se detalla más adelante en este capítulo.

### 5.2.2 Nivel del agente

El nivel del agente provee una especificación para implementar agentes. Los agentes controlan directamente a los recursos gerenciados y los hacen disponibles a las aplicaciones de gerenciamiento remotas. En general los agentes se encuentran ubicados en la misma máquina que los recursos gerenciados, aunque éste no es un requerimiento.
Un agente JMX contiene un servidor de MBeans o **MBean Server** que actúa como intermediario entre un conjunto de MBeans (que representan a los recursos gerenciados) y la aplicación de Gerenciamiento, uno o más servicios implementados también como MBeans y al menos un Adaptador de Protocolo (protocol adapter o PA) o Conector (conector o C).
Un agente JMX puede ejecutar en la misma máquina que mantiene los recursos monitoreados si tal máquina dispone de la máquina virtual java (JVM) o bien en otra máquina mediadora cuando el recurso no tiene capacidad para ejecutar una JVM. Los agentes JMX ejecutan sobre plataformas Java 2 (J2SE™ y J2ME™), versión 1.5 o superior.
Las aplicaciones de gerenciamiento tienen dos opciones para acceder a los MBeans de los agentes, esto es, pueden usar un **adaptador de protocolo o un conector**. Ambas componentes exponen las funcionalidades del agente a las aplicaciones gerenciamiento. La diferencia entre ellas radica básicamente, en cómo lo hacen. Los adaptadores de protocolo deben escuchar por mensajes entrantes que son construidos en un protocolo particular como HTTP o SNMP y están compuestos de una única componente que reside en el agente, mientras que los conectores están compuestos de dos componentes que residen una en el agente JMX y la otra en la aplicación de gerenciamiento; la comunicación ocurre entre estas dos componentes conectoras y de esta manera, ocultan el protocolo que está siendo usado en la comunicación entre el agente y el gerente.
Los protocolos y conectores permiten que una aplicación de gerenciamiento –ejecutando en una JVM diferente a la que está ejecutando el agente-, pueda:

- Recuperar y setear atributos a un MBean existente.
- Ejecutar operaciones sobre MBeans existentes.
- Instanciar y registrar a un MBean muevo.
- Registrarse para enviar y recibir notificaciones emitidas por MBeans.

Como puede observarse en la **Figura 5.1**, los **conectores (C)** son usados para comunicar un agente JMX con una aplicación JMX. Esta clase de comunicación involucra dos conectores uno del lado del servidor y otro del lado del cliente. Estas componentes llevan a cabo operaciones punto a punto en forma transparente usando un protocolo específico. Los servicios distribuidos del lado de la aplicación, proveen una interface remota del MBean Server, a través de la cual, la aplicación puede ejecutar las operaciones. Un conector es específico para un protocolo, pero la aplicación puede usar cualquier conector en forma indistinta, ya que ambos tienen la

misma interface remota. Por otro lado, los **adaptadores de protocolo (AP)** proveen a través de un protocolo una vista del agente JMX para su gerenciamiento. Adaptan las operaciones de los MBeans y del MBeans Server a una representación en un protocolo determinado como SNMP o HTML.
Tanto los conectores como los adaptadores usan servicios del MBean Server para llevar a cabo las operaciones que reciben para los MBean, y para reenviar notificaciones a las aplicaciones. Para que un agente pueda ser gerenciado remotamente, debe incluir al menos una de estas componentes. Sin embargo, la inclusión de más de uno de ellos, permitiría que el agente sea gerenciado por diferentes protocolos en forma simultánea.
La **Figura 5.2** muestra un agente JMX con un conector que le permite comunicarse con una aplicación JMX y dos adaptadores de protocolo que le proveen la comunicación tanto con un Navegador Web, como con una aplicación de Gerenciamiento existente en el mercado.

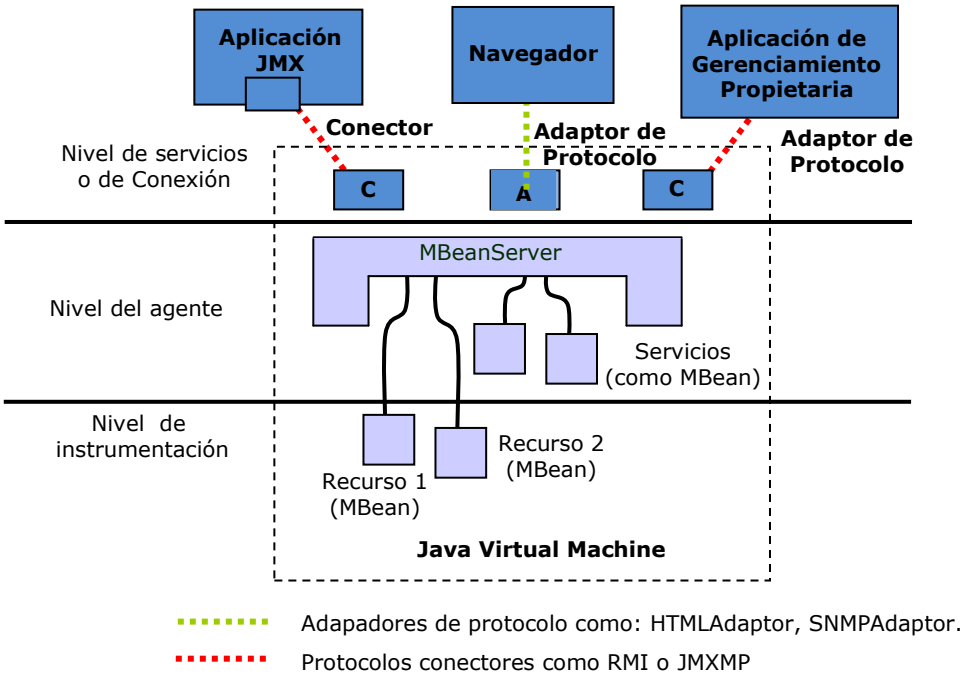



**Figura 5.2: Conexión entre agente y gerente**

### 5.2.3 Nivel de servicios distribuidos (Nivel del manager)

El nivel de servicios distribuidos proporciona las interfaces para implementar aplicaciones JMX. Este nivel define las interfaces de gestión y las componentes que pueden operar sobre los agentes. Estos componentes proporcionan una interface para que las aplicaciones de gerenciamiento puedan interactuar de forma transparente con los agentes JMX y gerenciar sus recursos a través de un conector. Este nivel facilita la consolidación de información proveniente de numerosos agentes JMX en vistas lógicas que facilitan el análisis de la información de gerenciamiento.

## 5.3 Componentes de la arquitectura JMX

Las componentes claves de la arquitectura JMX que define la especificación son las siguientes:

- **nivel de instrumentación** son los MBeans, el Modelo de Notificación y las clases metadata de los MBeans.
- **nivel del agente** son el MBean Server y los servicios del agente.
- **nivel de servicios distribuidos o conexión:** son los conectores o adaptadores de protocolos.

### 5.3.1 El nivel de instrumentación

**Los MBeans**

Los *beans* gerenciados o MBeans (del inglés Managed Beans) son objetos java que implementan una interface específica y cumplen con un determinado patrón de diseño. Estos requerimientos que deben cumplir los MBeans formalizan la representación de la ***interface de gerenciamiento*** del recurso en el MBean. La interface es la información que necesita una aplicación manager para poder operar con el recurso [JMXRes].
La interface de un MBean, está representada por:

- Los atributos que pueden ser accedidos.
- Las operaciones que pueden ser invocadas.
- Las notificaciones que pueden ser emitidas.
- Los constructores para la clase del MBean.

Los MBeans encapsulan los atributos y las operaciones a través de sus métodos públicos y cumplen con un patrón de diseño para exponerlos a las aplicaciones de gerenciamiento. Por ejemplo un atributo que es *read-only* tendrá solamente un método *setter*, mientras que uno *read-write* tendrá métodos *getter* y *setter*.

---

*Cualquier objeto que sea implementado como un MBean y registrado con un agente (en un MBean Server), podrá ser accedido desde afuera de la JVM donde ejecuta el agente.*

---

La arquitectura no impone ninguna restricción respecto de donde son guardados los MBeans compilados (.class). Ellos pueden ser guardados en cualquier ubicación local, especificada en el classpath de la JVM del agente o en un sitio remoto si se utiliza el cargador de clases ("*dynamic class loading*").
La principal división de los MBeans es en estándares y dinámicos. Los ***Standard MBean*** son los más simples para diseñar e implementar, su interface está compuesta por los nombres de sus métodos. Los ***Dynamic MBean*** deben implementar una interface específica, pero exponen su comportamiento en *run-time*, brindando mayor flexibilidad; se subdividen en ***Open MBeans, Model MBeans y MXBeans.***

**El Modelo de Notificación**

La especificación JMX define un modelo genérico de notificación basado en el Modelo de Eventos Java[4] [Hortsmann, C.]. Esta especificación define un modelo que permite a los MBeans, llamados **broadcasters**, propagar notificaciones. Las notificaciones pueden ser emitidas por las instancias MBean o por el Server MBean. Otros MBeans u

[4] En el modelo de delegación de eventos, la administración del evento es delegado desde la fuente del evento a uno o más objetos *listeners* encargados de atender ese evento.

objetos, interesados en eventos que produce el MBean, deben registrarse sobre uno o más MBeans, para ser notificados cuando un determinado evento ocurra. Estos son llamados **listeners**.
El modelo permite que un listener, se registre una única vez y reciba todas las notificaciones que ese MBean pueda emitir (hasta tanto no se quite esa registración). Además, un objeto listener puede registrarse en múltiples MBeans así como en un MBean se pueden registrar múltiples listeners.

## a. Listeners para notificación local

En los casos más simples, los **listeners** son objetos que se encuentran en la misma aplicación que los **broadcasters**. Los listeners son registrados invocando al método **addNotificationistener** sobre los bradcasters.
La **Figura 5.3**, ilustra la registración de 2 listeners locales al agente. L1 se registra directamente en el MBean broadcaster y L2 lo hace a través del MBean Server. El resultado es el mismo, ambos listeners reciben las mismas notificaciones, directamente desde el MBean broadcaster.

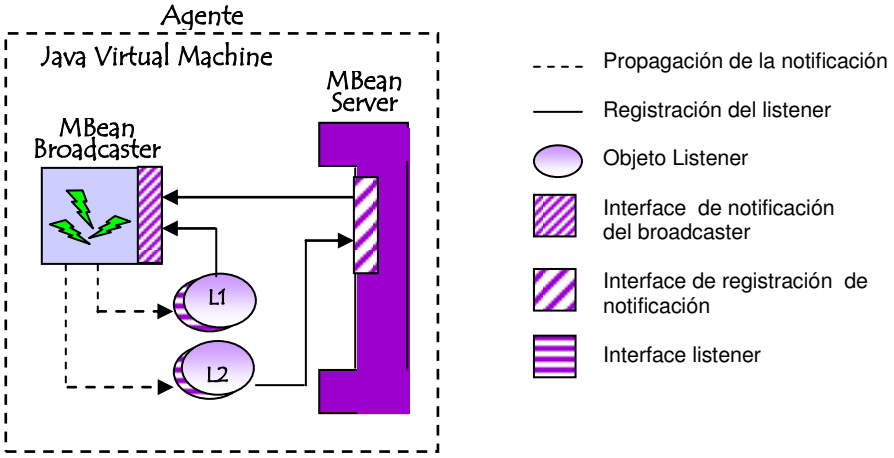



**Figura 5.3: *Listeners* locales del lado del Agente**

## b. Listeners para notificación remota

La interface cliente de un conector, provee el método **addNotificactionListener**, para permitir que las notificaciones sean recibidas en aplicaciones de gerenciamiento remotas.
Los *listener* no necesitan ninguna característica especial para transmitir remotamente. El conector transmite los requerimientos de registración y reenvía notificaciones de vuelta al *listener*. El proceso entero es transparente para el *listener* y para las componentes de gerenciamiento.
La **Figura 5.4** ilustra el comportamiento de los listeners remotos donde las componentes conectoras implementan un mecanismo complejo para registrar *listenters* remotos y reenviar notificaciones. Sin embargo, como ya describimos, el mecanismo usado para las notificaciones está basado en el modelo de delegación de eventos de java, por tanto, los MBeans broadcasters no pueden enviar notificaciones fuera de la JVM.
*¿Cómo se resuelve esta situación?*
El servidor conector instancia un *listener* local, encargado de recibir todas las notificaciones, ponerlas en caché y enviarlas a la aplicación manager en bloque, de

forma de evitar la saturación en la capa de comunicación cuando hay muchas notificaciones simultáneas.
Este mecanismo está presente en los nuevos protocolos RMI[5] y JMXMP[6], donde las notificaciones son *pulled* periódicamente en los requerimientos del cliente. El mecanismo de *pull* es usado para agrupar notificaciones y reducir el uso del ancho de banda utilizado para notificar eventos. El conector cliente, actúa como un *broadcaster* enviando las notificaciones a los *listenters* interesados. La parte III de la especificación JMX [Ref. 5-2] contiene más información sobre estos protocolos que permiten acceder remotamente a un JMX MBean Server.

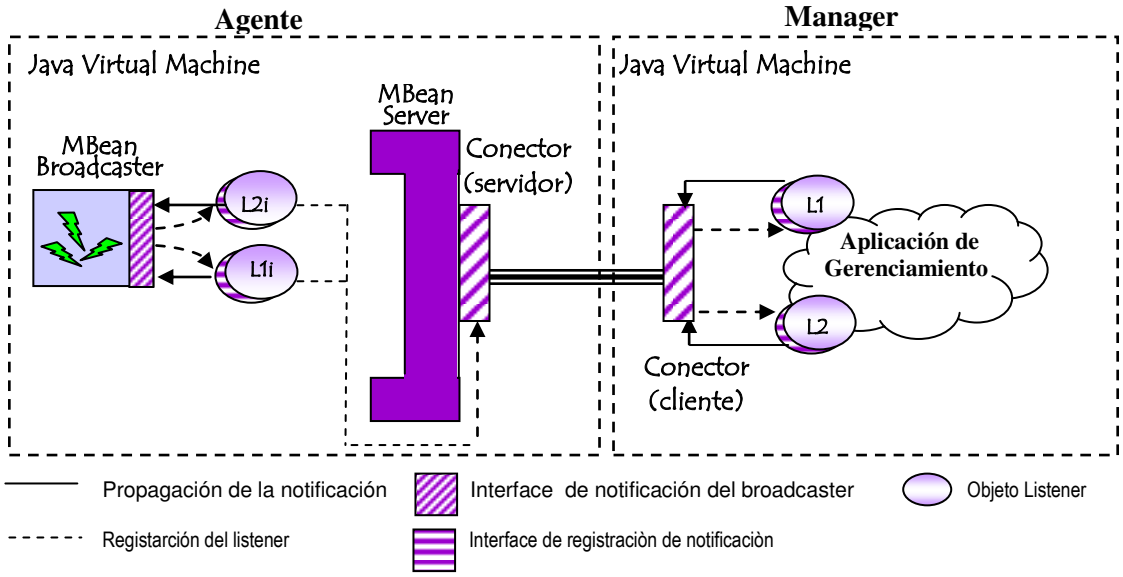



**Figura 5.4: *Listeners* remotos del lado del Manager**

### 5.3.2 El nivel del agente JMX

Las componentes claves en el nivel del agente son el *MBean Server*, el cual funciona como un repositorio para los objetos del nivel de instrumentación y los servicios que le permiten a un agente JMX incorporar inteligencia para más autonomía y mejor *performance*.

#### El MBeanServer

Un MBean Server es un repositorio de objetos, que además los expone para operaciones de gerenciamiento. Si bien, cualquier objeto registrado con un *MBean Server* se vuelve visible para las aplicaciones de gerenciamiento, el MBean Server solamente expone interfaces de gerenciamiento de los MBean que mantiene y nunca la referencia del objeto.
Los MBeans pueden ser instanciados y registrados por:

- Otro MBean
- El propio agente

5 El java Remote Method Invocation (RMI) permite que un objeto ejecutando en una JVM, invoque métodos de un objeto corriendo en otra JVM. RMI provee la comunicación remota entre programas escritos en java.
6 Un conector hace a un MBeanServer accesibe remotamente, a través de un protocolo. La API JMX para comunicación remota, define un conector estándar, el **conector RMI** (**RMI Connector**), el cual provee acceso remoto a un MBeanServer a través de RMI. Esta API también define un protocolo opcional llamado **JMXMP (JMX Message Protocol)** con el mismo propósito.

- La aplicación de gerenciamiento remota.

Cuando se registra un MBean se lo debe hacer por ***nombre,*** el cual debe ser único. Una aplicación utiliza el nombre del objeto para identificarlo y ejecutar las operaciones de gerenciamiento sobre él.
En la **Figura 5.5** se observa un agente, que tiene un MBean Server que contiene registrados servicios y recursos que pueden ser expuestos a aplicaciones de gerenciamiento mediante un conector y/o un adaptador.

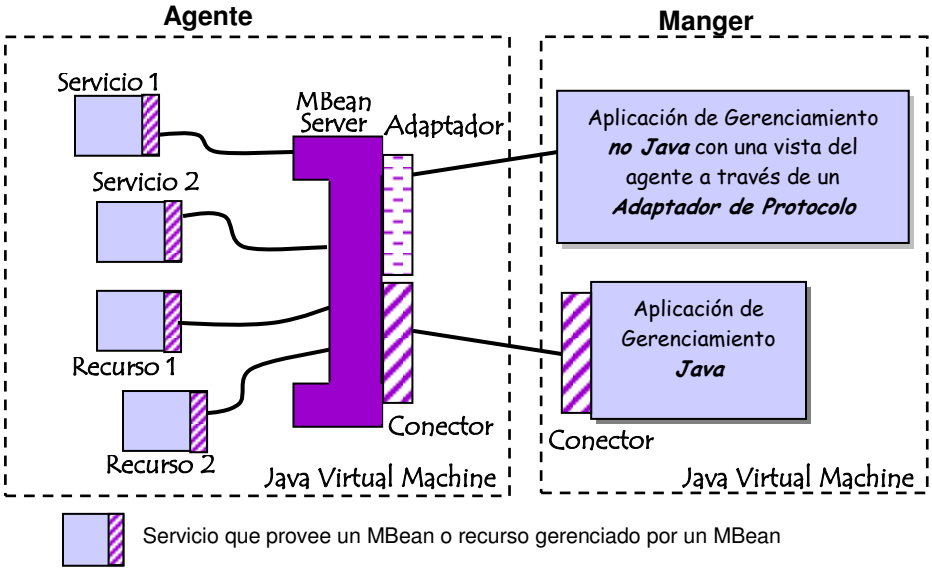



**Figura 5.5: Arquitectura de un agente JMX**

El MBean Server depende de los adaptadores de protocolo o de los conectores para hacer a un agente accesible desde las aplicaciones de gerenciamiento (que están ejecutando en otra máquina o en el caso de las aplicaciones JAVA, en otra JVM diferente a la del agente). Cada adaptador provee una vista a través de un protocolo específico de todos los MBeans registrados en el MBean Server. Por ejemplo, un adaptador HTML (*HTML adaptor)*, puede desplegar un MBean en un navegador web o un adaptador SNMP *(SNMP adaptor)*, puede exponer MBeans especiales que representan una MIB SNMP y responder a requerimientos del protocolo SNMP. Un conector provee una interface que facilita la comunicación entre el manager y el agente. Cada conector provee la misma interface remota, a través de un protocolo diferente. Una aplicación de gerenciamiento usa esta interface para conectarse a un agente, y lo hace de manera transparente, sin importar el protocolo.
Los conectores y los adaptadores hacen disponibles para las aplicaciones remotas las operaciones del MBean Server. Para que un agente pueda ser manejado, debe incluir al menos un adaptador o un conector, pero también puede incluir más de uno, permitiendo que el mismo sea manejado por múltiples managers y a través de múltiples protocolos.

### 5.3.3 El nivel de servicios distribuidos

Este nivel está compuesto principalmente por los Conectores y los Adaptadores de protocolo. Provee una interface para que aplicaciones de gerenciamiento puedan

interactuar en forma transparente con un agente y los recursos que él gerencia. Los conectores y los adaptadores, exponen una vista de un agente JMX y sus MBeans, mapeando su significado semántico en un protocolo, esto es, permiten obtener una vista muy simple del agente con un protocolo como HTML o con un protocolo de más bajo nivel como SNMP. Asimismo, usando los conectores se puede desarrollar una aplicación de gerenciamiento visual usando clases de la API de JAVA para crear interfaces de usuario gráficas y poder interactuar con los agentes de manera visual.

## 5.4 Arquitectura de Cafeto

CAFETO, nace como una aplicación de gerenciamiento *basada en web* y *colaborativa* para monitoreo de agentes SNMP. Inicialmente se puso énfasis en la definición de una aplicación manager con una interfaz de usuario de alta calidad y colaborativa que le facilite a los administradores cooperar en la resolución de problemas. Para ello se implementó un primer prototipo con una interfaz de usuarios construida con componentes de JFC[7] [Geary, D.] con el que se realizaron las primeras pruebas de conectividad con un agente SNMP. La **Figura 5.6** muestra la primera y muy simple arquitectura Cafeto.

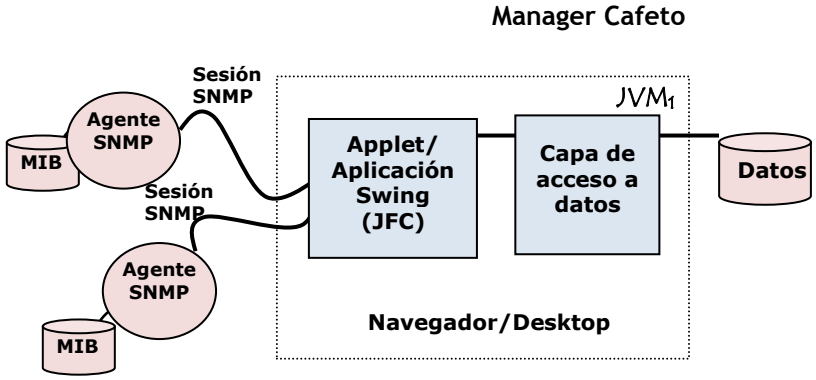



**Figura 5.6: Arquitectura inicial de Cafeto**

En este prototipo, el manager Cafeto puede iniciarse como un applet desde cualquier navegador o como una aplicación descargada usando Java Web Start[8]. Este manager establece conexiones punto a punto con cada uno de los agentes SNMP que monitorea. Ambas modalidades cuentan con la misma interfaz de usuario gráfica a partir de la cual se puede construir visualmente una red, ejecutar consultas por medio de menúes y obervar los resultados.
Si bien este prototipo permitió realizar los primeros testeos con la API, no mejoró las limitaciones del protocolo SNMP: *agentes sin inteligencia y tráfico constante de gerencimiento en la red*. Considerando las limitaciones de este prototipo, se comenzó

[7] Java Foundation Classes (JFC): Es un conjunto de tecnologías para escribir applets y aplicaciones de escritorio con una interfaz de usuario de alta calidad. Está formada por un conjunto de APIs que permiten lograr intefaces visuales con imágenes, sonido, gráficos 2D, drag&drop, y con una misma apariencia a pesar de las distintas plataformas de ejecución.
[8] Java Web Start (JWS): **Java Web Start** es la implementación de referencia de la especificación JNLP (Java Networking Launching Protocol) y está desarrollada por Sun Microsystems. Permite arrancar aplicaciones Java que están en un servidor web de aplicaciones comprobando previamente, si el cliente tiene la versión actualizada de dicha aplicación -si no es así, descargará la última versión antes de ejecutar-. El arranque de dichas aplicaciones puede ser efectuado mediante enlaces en una página web o bien a través de enlaces en el escritorio cliente. JWS viene incluido en el JRE (Java Runtime Environment).

a pensar en implementar un agente con más inteligencia, que pudiera monitorear independientemente del manager y que realice notificaciones al manager únicamente ante situaciones extremas -eventos anormales *y/o* cambios en los cuales el manager se interesó-. Para la implementación de este agente se continuó testendo la API JMX descripta en al capítulo anterior y se comenzó a diseñar el agente CAFETO, dando lugar a una nueva arquitectura ilustrada en la **Figura 5.7.**
En esta nueva arquitectura el manager se comunica con un agente java y a su vez, éste se encarga de monitorear a un agente SNMP. El agente JAVA también podría comunicarse con navegadores web y con otras aplicaciones de gerenciamiento.

*¿Cuál es el beneficio de esta arquitectura?*
El agente envía notificaciones a los managers únicamente ante situaciones especiales, disminuyendo el tráfico de la red. Asimismo, para la implementación de este agente se dispone de toda la API de JAVA con lo cual se puede dotar al agente de cualquier funcionalidad e inteligencia.

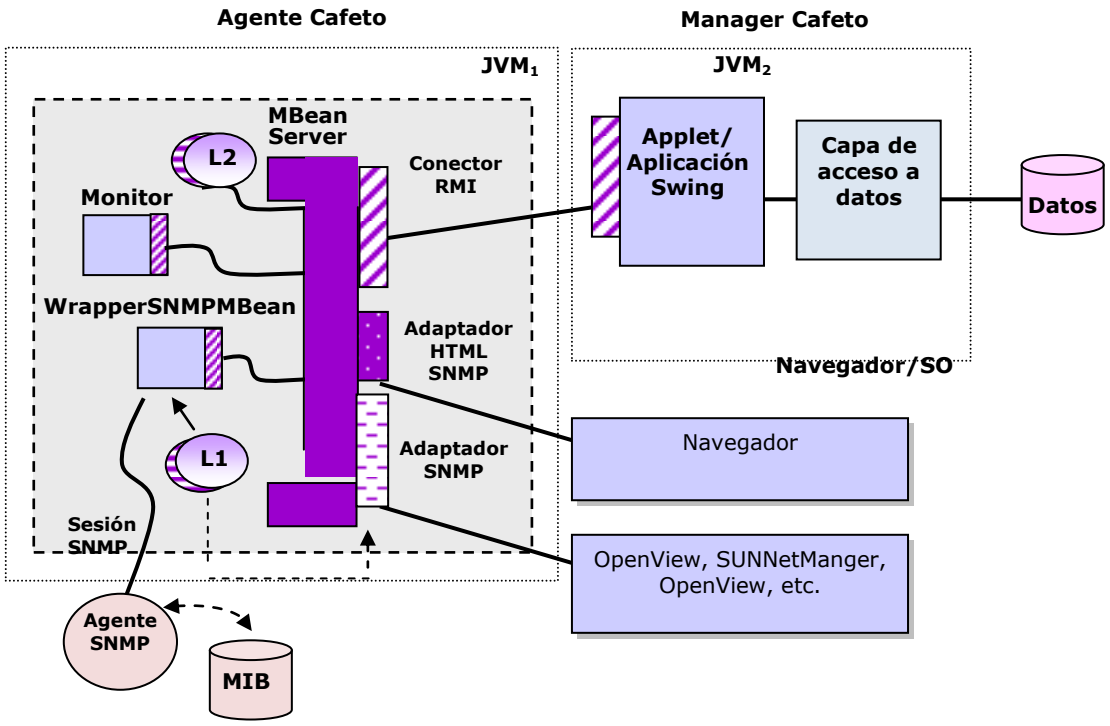



**Figura 5.7: Arquitectura de Cafeto modificada**

Si bien estos agentes ayudan a reducir el tráfico de monitoreo y además, podrían contener cualquier código inteligente, son viables sólo para monitorear recursos que puedan ejecutar código java, y es sabido que en muchos dispositivos, como es el caso de routers cisco -donde se prioriza la *performance*- no se permite la ejecución de código java. Además de esta limitación relacionada con los tipos de dispositivos, cuando el número de agentes SNMP a adminitrar es muy grande, esta arquitectura impone la necesidad de instalar un agente JAVA en cada dispositivo a monitorear, lo que complica la instalación y mantenimiento de los agentes JAVA en cada uno de los dispoitivos monitoreados. Si la arquitectura continene muchos dispositivos -con SNMP- destinar una instancia del agente JAVA para manejar cada agente SNMP es complicado. Para integrar a los dispositivos sin capacidad java y para disminuir la

complejidad planteada, se propone una arquitectura donde un agente JAVA concentre y envíe a la aplicación de gerenciamiento información de monitoreo correspondiente *a un conjunto de agentes SNMP*. Esta propuesta simplifica la administración de los dispositivos, habilita el monitoreo de dispositivos sin capacidad JAVA, disminuye considerablemente el tráfico de la red y como complemento, no interfiere en las políticas de seguridad de la red monitoreada por el agente JAVA.
La **Figura 5.8**. muestra la arquitectura definitiva propuesta en esta tesis donde se puede observar que un único agente JAVA monitorea a un conjunto de agentes SNMP. El agente CAFETO funciona como un proxy entre los agentes SNMP y la aplicación de gerenciamiento. En la figura también pueden observarse varios MBeans y objetos *listeners* registrados en el MBean Server, un conector RMI para que el agente pueda ser accedido por la aplicacción cliente JAVA, un adaptador SNMP para permitir también el acceso a una aplicación de gerenciamiento propietaria y un adaptador HTML para poder acceder al agente desde un navegador web.

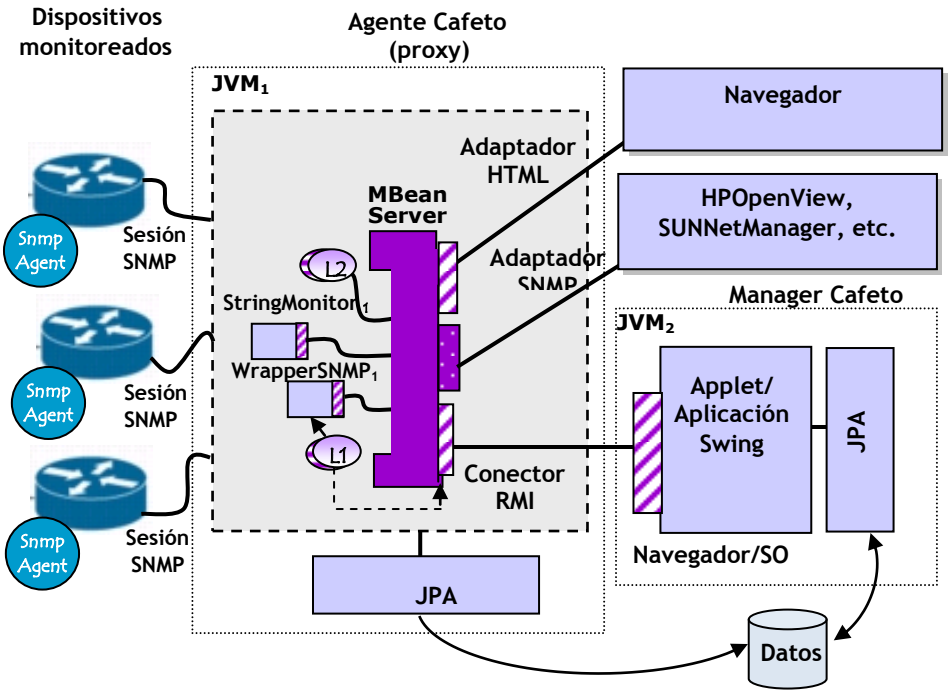



**Figura 5.8: Arquitectura definitiva de CAFETO**

A continuación se describe con más detalles la arquitectura y la funcionalidad del agente JAVA y de la aplicación de gerenciamiento, así como también los protocolos de comunicación que facilitan el intercambio de información entre ellos.

### 5.4.1 El agente Cafeto

Los servicios que provee el agente Cafeto son representados, según la especificación JMX, como MBeans. Para testear las capacidades de la API, se han definido algunos MBeans a manera de ejemplo, sin embargo podrían incorporarse a este agente JAVA, más y variados MBeans.
Se ha definido un MBean llamado **`WrapperSNMP`** para brindar un *servicio de acceso a SNMP;* este MBean funciona como un *wrapper* del agente SNMP, brinda los servicios que provee un agente SNMP y además, es posible agregar cualquier funcionalidad

implementada con java. Así mismo se han definido otros MBeans denominados **`AtributoString`**, **`AtributoCounter`** y **`AtributoGauge`** para brindar *servicio de gerenciamiento de atributos de las MIBs***;** estos MBean permiten controlar desde JAVA el cambio de los valores de los atributos de diferentes tipos de la MIB.
Sin esta funcionalidad, deberíamos establecer una conexión desde la aplicación manager, a cada agente SNMP, y hacer *polling*s constantemente a este agente, hasta determinar cualquier cambio de valor en uno o varios atributos o si algún atributo o combinación de ellos, alcanzaron un valor tope. Con este MBean, NO hay tráfico en la red, ya que el agente Cafeto es el que hace los *polling*s -según se definió usando el manager Cafeto- y le avisa a la aplicación de gerenciamiento únicamente ante una situación especial. Para lograr esto, se hace uso de distintos monitores, que se describen en el próximo capítulo, a los cuales se les delega la tarea de monitoreo.
Este agente puede ser accedido remotamente a través de diferentes adaptadores de protocolo y conectores, estos objetos, al igual que los servicios recientemente descriptos, son parte del agente. Como puede observarse en la **Figura 5.8**, el agente puede ser accedido por un manager Cafeto, por un navegador web o por cualquier aplicación de gerenciamiento no Java, obteniendo respectivamente una vista del agente a través de un adaptador de protocolo. El agente cuenta con dos adaptadores de protocolo y un conector que se describen a continuación.
El ***Adaptador de protocolo HTML*** actúa como un servidor HTML permitiendo que cualquier navegador consulte los MBeans del agente a través del protocolo HTTP. En el caso de cafeto, fue incorporado en el agente especialmente como una herramienta de testeo, ya que de esta manera no se pueden aprovechar las funcionalidades de la aplicación de gerenciamiento JAVA. Cuando el adapatador de protocolo HTML es arrancado, crea un *socket* TCP/IP, escucha por conexiones de managers al agente y espera por pedidos de requerimientos. Este adaptador está implementado como un MBean dinámico.

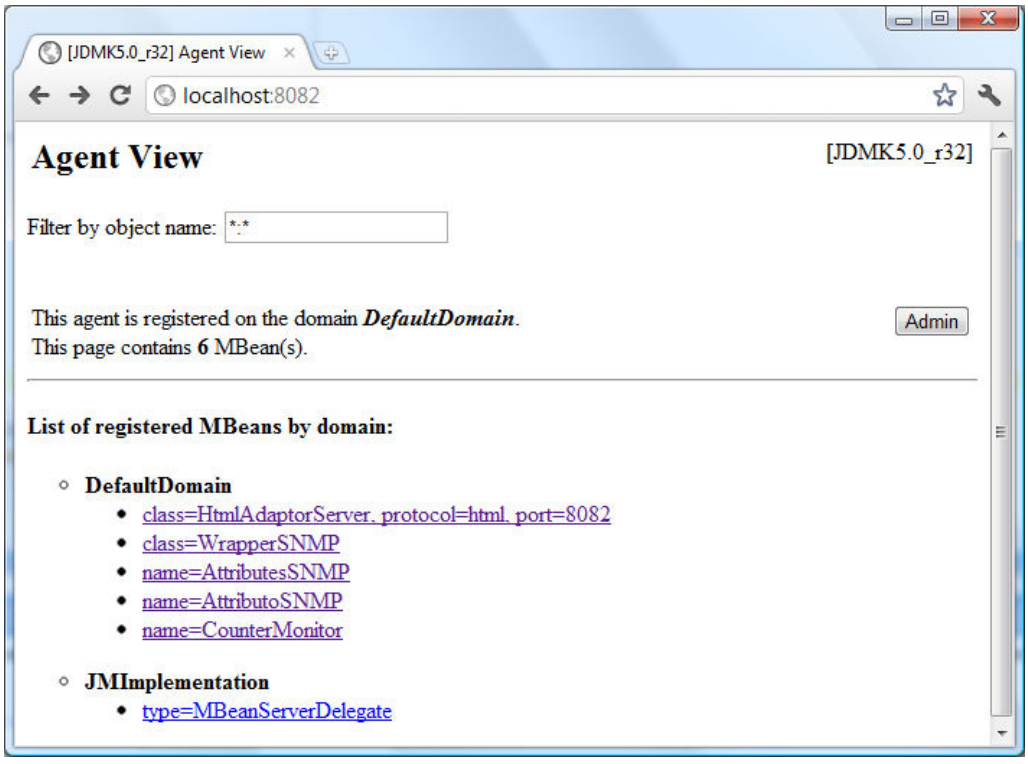



**Figura 5.9: Visualización de los MBeans del Agente CAFETO**

La **Figura 5.9**. muestra la interaz de usuario de este adaptador. Presionando los links a los MBeans se puede acceder a sus propiedades y valores. La información de cada MBean es deplegada en forma textual em una página HTML.

El ***Adaptador de protocolo SNMP*** no interactúa con MBeans de la misma manera que lo hacen los otros adaptadores y conectores. Esto es porque los datos de SNMP están en MIBs, y únicamante MBeans que representan a las MIBs, pueden ser manejadas a través de SNMP. El adaptador SNMP no interactúa con MBeans de una MIB a través del servidor MBean, sino que ellos deben ser explícitamente ligados a la instancia del adaptador SNMP. El adaptador de protocolo SNMP maneja solo MBeans derivados de MIBs, sin embargo, otros protocolos podrán acceder a estos MIBs.

El ***Conector RMI*** faclilita las comunicaciones remotas entre el agente y los gerentes. Existen dos implementaciones de este conector usando los protocolos de transporte estándares RMI: el Java Remote Method Protocol (JRMP) o el Internet Inter-ORB (IIOP). El conector RMI sobre JRMP, provee un mecansimo simple para manejar seguridad y autenticar la conexión entre un cliente y un server. El conector usado provee una conexión punto a punto entre el agente cafeto y un manager cafeto. El agente tiene registrado un conector Server y la aplicación de gerenciamiento hace uso de la parte cliente del conector, para establecer la comunicación. El conector cliente, expone una versión remota de la interface del MBeanServer. Cada conector cliente representa un agente al cual el manager quiere conectarse y para hacerlo, debe conocer el host y el port RMI. Por otro lado, el conector server, esto es, el que está en el agente, escucha por requerimientos de conexiones de aplicaciones, crea una conexión para cada requerimiento y replica a todos simultáneamente.

En Cafeto se usó un conector desarrollado por SUN, de las APIs de JDMK. La **Figura 5.10** ilustra la comunicación entre los managers y los agentes de Cafeto usando RMI.

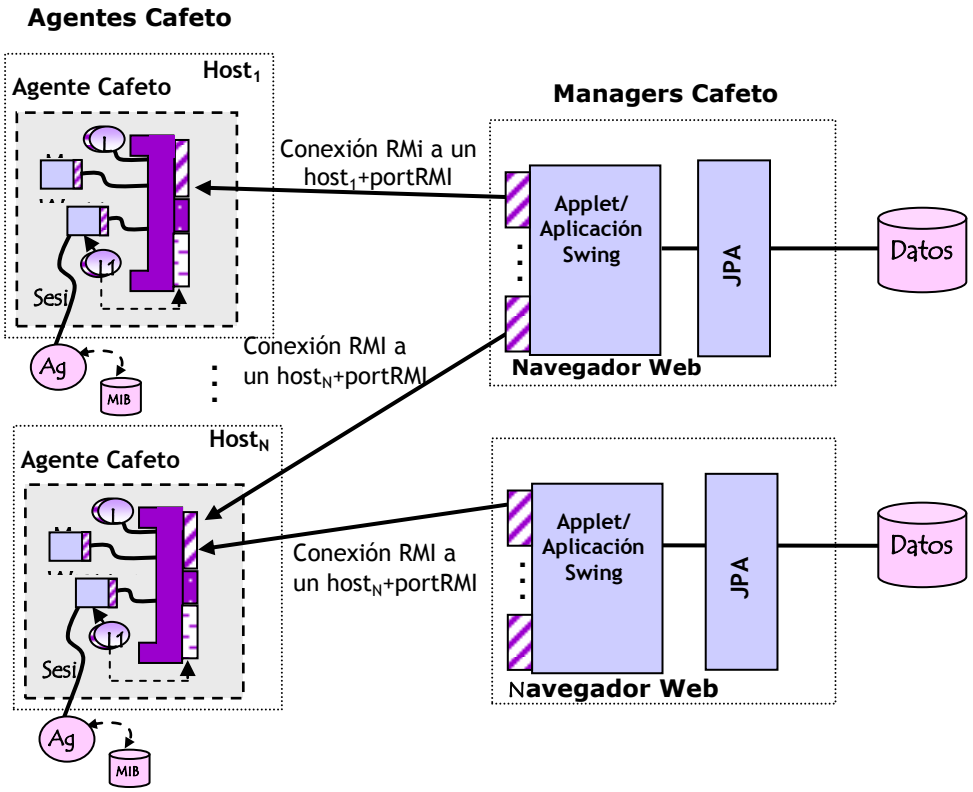



**Figura 5.10: Comunicación entre Agentes y Managers de Cafeto**

Este adaptador de protocolo HTML no se ajusta muy bien si se tienen muchos MBeans, tampoco para tipos de datos complejos.

Finalmente, también forman parte del agente los objetos *listeners*. Para este prototipo se han implementado a modo de ejemplo dos un *listeners*: uno llamado **`ChangeValueListener`** para recibir notificaciones emitidas por un MBean cuando el valor de uno de sus atributos cambia y otro denominado **`ThreadholdListener`** para ser registrado en un Monitor y notificado cuando el valor o los valores observados por el mencionado monitor alcanzan un valor umbral o *threadhold*.
Los detalles de implementación de la comunicación entre los agentes y los managers y la lógica de los objetos *listeners* están descriptos en el Anexo A.

### 5.4.2 El manager Cafeto

Uno de los objetivos con los que se emprendió esta tesis fue el desarrollo de una aplicación para monitoreo con una interfaz de usuario de alta calidad, colaborativa y basada en web. Por este motivo se decidió utilizar *Java Foundation Classes* [Geary, D; a y b] para la construcción de la Interfaz de Usuario de la aplicación manager. Como ya se ha mencionado, la aplicación manager Cafeto, puede ser usada en dos modalidades: como un *Applet Swing* o como una aplicación de escritorio *Swing*. Ambas tienen la misma GUI, la diferencia está en la forma de trabajo de cada una. En el caso del applet, el administrador necesita estar conectado a la web, y mantenerse en una página web a través de la cual se accede al applet. En el caso de la aplicación, el administrador debe acceder una vez a la página principal de Cafeto, presionar un link y automáticamente el JavaWebStart descargará la aplicación para trabajar en una ventana independiente a la del navegador.
Por otro lado, para la comunicación del gerente con los agentes se hace uso del nivel de servicios distribuidos de JMX, básicamente de los conectores y del adaptador de protocolo SNMP.
La arquitectura del manager puede observarse en el siguiente gráfico:

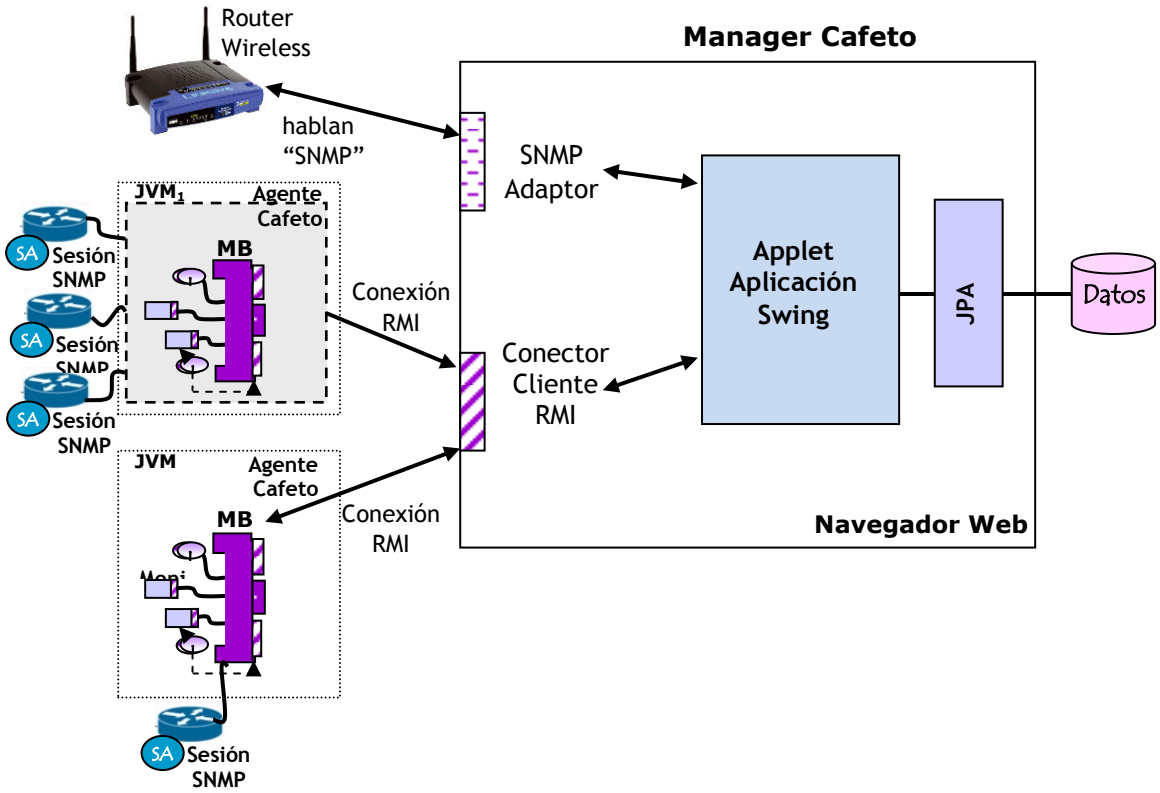



**Figura 5.11: Arquitectura del Manager Cafeto**

### Comunicación con el agente: conectores y adaptadores de protocolo

La especificación JMX, no define ninguna arquitectura para la aplicación de gerenciamiento, sólo brindan los conectores para facilitar la comunicación. El conector cliente, es decir, el que se encuentra en la aplicación de gerenciamiento Cafeto, provee una interface remota del MBean Server del agente Cafeto, a través de la cual, la aplicación puede ejecutar las operaciones. Un conector es específico para un protocolo, pero la aplicación puede usar cualquier conector en forma indistinta, ya que ambos tienen la misma interface remota.
En el caso de que el recurso a ser monitoreado disponga de la JVM, que es la mayoría de los casos, puede ejecutarse el agente JMX, el cual se comunicará con la aplicación Cafeto. Cuando desde la aplicación de gerenciamiento se quiere monitorear un dispositivo que no soporte java, pero si dispone de un agente SNMP, el manager puede comunicarse directamente usando una Adaptador de protocolo SNMP. En este último caso, la aplicación de gerenciamiento, hablaría SNMP con el agente SNMP, perdiendo los beneficios provistos por el agente JMX.

## 5.4.3 Comunicación con la Capa de Datos

Para la persistencia de la información de gerenciamiento se utilizó Java Persistence API o JPA [Bauer C.], la interface para gerenciamiento de persistencia y mapeo objecto/relacional estándar de la plataforma Java Enterprise Edition 5.0 (JEE 5). JPA es soportada por la mayoría de los contenedores J2EE java, sin embargo, el nuevo estándar establece que el motor JPA debe ser capaz de correr fuera de un ambiente de ejecución EJB 3.0. Por lo tanto cualquier aplicación J2SE, y en este caso el agente y el gerente pueden hacer uso de JPA sin necesidad de un contenedor EJB 3.0 o servidor Java EE.
Para acceder a la base de datos, además se implementó una capa de acceso a datos, comúnmente conocida como DAO. Esta capa provee una separación de lo referente a la persistencia de objetos y a la lógica de acceso a los datos, de cualquier mecanismo de persistencia o API particular, permitiendo flexibilidad para cambiar el mecanismo de persistencia sin necesidad de cambios en la lógica de la aplicación que interactúa con la DAO. Actualmente la persistencia está implementada con JPA, pero podría modificarse para usar directamente JDBC u otra API sin cambios significativos.

## 5.4.4 Beneficios alcanzados con la arquitectura propuesta

Aprovechando la funcionalidad provista por el protocolo SNMP y la popularidad del mismo en el área de gerenciamiento de redes, la arquitectura propuesta fue diseñada para monitorear recursos con capacidades SNMP. El agente JAVA implementado, corazón de la arquitectura propuesta, puede ser visto como un *wrapper* de agentes SNMP, que logra reducir la carga de la red, evitando que la aplicación de gerenciamiento hable con cada agente SNMP y habilitando al agente JAVA para hacerlo. De esta manera el agente JAVA envía notificaciones a la aplicación de gerenciamiento, únicamente cuando es necesario.
Asimismo se aprovecha la disponibilidad de SNMP en todos los dispositivos, y a su vez, se extiende su funcionalidad. La funcionalidad de los agentes SNMP es limitada, pero la de un agente JMX como se demuestra en esta tesis, puede cubrir todas las

necesidades de gerenciamiento. El agente JAVA puede monitorear recursos, detectar si determinados valores alcanzan un límite y en respuesta a eso, avisarle a la aplicación de gerenciamiento mediante efectos visuales, enviar mensajes a los administradores o realizar cualquier tipo de acción. Se logró así un agente con autonomía y más inteligencia.
Con esta arquitectura también se puede incrementar la funcionalidad del agente en *run-time ya que* se pueden agregar nuevas MIBs y servicios desde el manager, sin detener el agente. Asimismo el agente java implementado puede ser monitoreado no sólo por la aplicación de gerenciamiento JMX sino también por cualquier aplicación de gerenciamiento SNMP (OpenView, SunNetManager, etc.).
Finalmente, la arquitectura final propuesta requiere de una única instalación del agente JAVA en una máquina servidora a partir de la cual se puede monitorear un conjunto de dispositivos internos con capacidad SNMP. Este modelo no sólo simplifica la instalación sino que favorece a la seguridad de la red.

### 5.4.5 Síntesis

En este capítulo se describe la arquitectura de Cafeto, la cual está basada en Java y destinada a mejorar el monitoreo de redes basadas en SNMP. La arquitectura que se presenta consta de un manager compatible con Java Management Extensions (JMX) que se comunica con agentes JMX, los cuales funcionan como mediadores entre el manager y un conjunto de agentes SNMP.
La especificación JMX define una arquitectura para gerenciamiento y un conjunto de APIs que describen las componentes de esta arquitectura. Tal especificación define una arquitectura de tres niveles, que cubre funcionalidades para implementar agentes que puedan comunicarse con gerentes. Los niveles de esta arquitectura son: nivel de instrumentación, nivel del agente y nivel de servicios distribuidos. Adicionalmente provee un conjunto de APIs para protocolos de gerenciamiento existentes, entre ellas para SNMP que fue utilizada en esta tesis.
La arquitectura propuesta está basada en JMX y permite extender las capacidades de los agentes SNMP, reducir la carga de la red y favorecer a la administración y seguridad de la misma.

# Capítulo 6

## CAFETO: Un prototipo basado en web para Gerenciamiento de Redes de Datos

### 6.1 Introducción

En este capítulo se describe a CAFETO, un prototipo que implementa en JAVA la arquitectura de gerenciamiento propuesta en el capítulo anterior. Para definir las funcionalidades de esta aplicación agente/gerente se tomó como marco de referencia el modelo de gerenciamiento FACPS -del inglés *Fault, Configuration, Accounting, Performance, Security* [Clemm, A.] [Leinwand A, Conroy, K] que organiza a las funciones de gerenciamiento en cinco áreas básicas. El capítulo comienza con una descripción de cada una de estas áreas mostrando las funcionalidades propuestas por CAFETO para alguna de ellas. Luego se describen las características de CAFETO desde el punto de vista de la interfaz de usuario, roles de usuarios y finalmente se concluye con los beneficios logrados a partir de la arquitectura planteada y del prototipo implementado.

### 6.2 Características de CAFETO desde el punto de vista del Gerenciamiento de Redes

Con el objetivo de definir mejor el alcance que tiene una administración eficiente de redes, comúnmente se suele agrupar a las funciones de gerenciamiento en cinco áreas funcionales o categorías a saber: manejo de fallas, configuración, contabilidad, *performance* y seguridad. A continuación se describen cada una de estas categorías funcionales o áreas del FCAPS, haciendo especial énfasis en el soporte que brinda CAFETO para gerenciar algunas de ellas.

#### F de Fault

El ***manejo de fallas*** es el proceso de ubicar problemas o fallas en la red. Esto involucra descubrir el problema ocurrido, aislarlo y si es posible solucionarlo. Está relacionado con el *monitoreo* de la red para asegurarse de que cada cosa esté funcionando correctamente o de lo contrario, para reaccionar ante fallas.

El manejo de fallas podría incluir:

- Monitoreo de la red incluyendo procesamiento de alarmas
- Diagnóstico de fallas y análisis de las causas de las fallas
- Mantenimiento de un historial de alarmas

El monitoreo de una red incluye funciones que permiten a un administrador determinar si la red está funcionando como se esperaba, realizar un seguimiento y visualizar el estado de la red. Esta funcionalidad es fundamental para reconocer y reaccionar ante condiciones de falla en la red a medida que ocurren.

Uno de los aspectos más interesantes del monitoreo es la generación y gerenciamiento de las alarmas. Las alarmas son mensajes no solicitados provenientes de la red, que indican que algún evento "especial" ha ocurrido y que en algunos casos requiere la intervención de un administrador. La gestión de alarmas es un área muy amplia, incluye muchas funciones básicas, que van desde la recepción y la visualización de las alarmas hasta funciones más avanzadas que implican el procesamiento de las mismas para realizar tareas de filtrado y correlación.

CAFETO define un sistema de alarmas ampliamente configurable que incluye la visualización actualizada de las fallas en el mapa de la red, el envío de mensajes a cuentas de correo electrónico o a dispositivos móviles y el registro de las fallas -con su motivo y acciones efectuadas para su resolución- en un libro histórico de fallas.

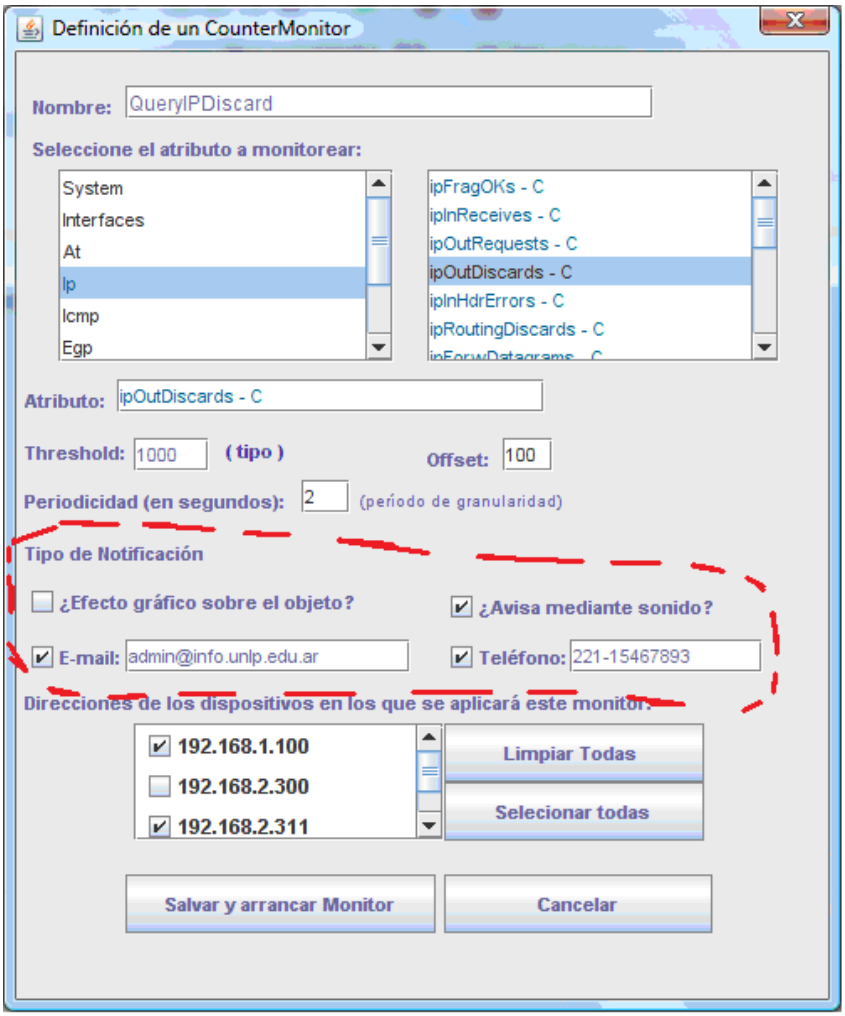



**Figura 6.1: Configuración de alarmas desde la Aplicación de Gerenciamiento**

La **Figura 6.1** muestra una de las ventanas de diálogo que brinda CAFETO donde se puede configurar el *tipo de notificación* que se desea ante la ocurrencia de una situación anormal. Esta ventana que corresponde a la aplicación de gerenciamiento provoca la activación de un monitor en el agente remoto para que el mismo envíe alarmas a la aplicación de gerenciamiento ante de detección de una falla.

Todas las fallas recibidas por la aplicación de gerenciamiento son registradas en una base de datos para la construcción de un *historial de alarmas*. La registración de la falla es automática, los datos que se registran son: la causa –datos y condiciones registradas-, la fecha, la hora y el dispositivo en el cual se generó la anomalía. El administrador de la red, además de ver las fallas registradas puede incorporar un detalle con los pasos seguidos para solucionar el problema y encuadrar a la falla en un nivel de gravedad. Esta información puede ser consultada por fecha, tipo de error, dispositivo, etc., y puede ser útil para la resolución de problemas similares en un futuro.

El manejo de fallas también tiene un impacto en la interfaz de usuario de la aplicación de gerenciamiento. CAFETO provee un panel donde se *visualiza la topología de la red* utilizando un esquema de colores y efectos especiales para indicar el estado actual de cada dispositivo. La tabla de la **Figura 6.2** describe el uso de diferentes representaciones visuales para los distintos estados.

| Estado del elemento gerenciado | Efecto visual |
|---|---|
| **Dispositivo sin falla** | su ícono normal |
| **Dispositivo con posible falla** | su ícono titilando |
| **Dispositivo con falla detectada** | su ícono coloreado en rojo y titilando |
| **Dispositivo que fue levantado después de un error** | su ícono coloreado en amarillo |
| **Dispositivo que no responde** | su ícono acompañado por el trazo "?" |
| **Dispositivo caído/inalcanzable** | su ícono tachado con un trazo de cruz |

**Figura 6.2: Tabla de efectos visuales usadas por CAFETO**

## C de Configuration

La ***configuración***, es el proceso de encontrar y setear dispositivos críticos. El área de configuración abarca funcionalidades que permiten cambiar valores, como dirección IP, tablas de ruteo, etc. de los equipos de una red. Incluye una configuración inicial para que el dispositivo esté correctamente conectado a la red, así como también cambios de configuración en curso.

La configuración no sólo puede ser hecha por los administradores, también es posible que el sistema haga descubrimiento automático de los dispositivos de la red. Esta función puede ser útil cuando:

- Los registros de inventario no son exactos
- Se cambian cosas en la red y no se registran los cambios correctamente
- Se comienza a utilizar una aplicación de gerenciamiento y se prefiere evitar la carga manual de cada elemento de la red

La aplicación de gerenciamiento CAFETO permite diseñar en forma interactiva la topología de la red a monitorear. Los administradores pueden por medio de la técnica de manipulación directa acomodar los elementos a gerenciar sobre la pantalla, logrando una representación lógica de la red a administrar. Esta representación permite visualizar la topología y el estado la red –mediante algunos de los efectos especiales descriptos en la tabla de la Figura 6.2-. Los usuarios administradores pueden agregar nuevas componentes, modificar los atributos de las componentes existentes y eventualmente, dar de baja a elementos que ya no desea gerenciar.

La **Figura 6.3** muestra la pantalla principal de CAFETO, desde donde se puede construir la topología de la red.

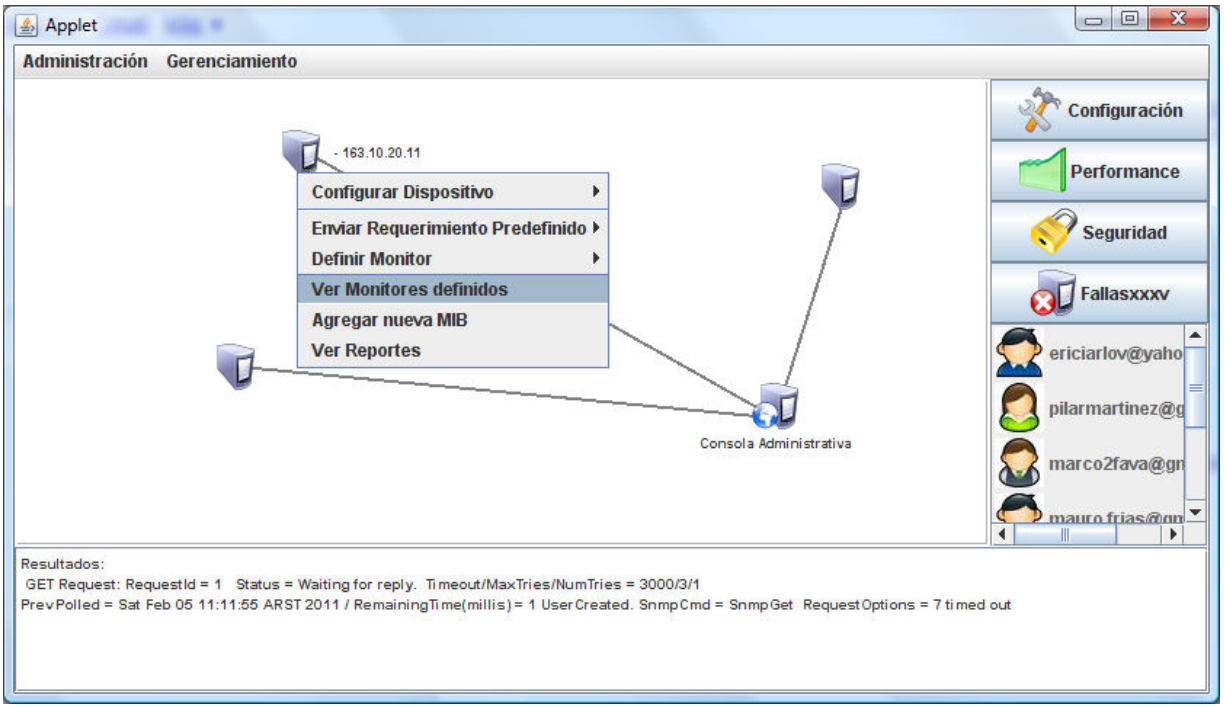



**Figura 6.3: Configuración gráfica de la red**

La API JMX también provee un paquete que facilita el *autodiscovering*, de manera de que el agente podría usar este servicio para obtener la configuración automática inicial de la red que va a monitorear. Este prototipo no hace uso de ese servicio, en reemplazo, cada agente JAVA cuando arranca lee desde un archivo de propiedades las direcciones IP de todos los dispositivos SNMP que pertenecen a la red local y que se quiere monitorear.

## A de Accounting

***Accounting*** involucra medir la utilización de los recursos de la red, para establecer métricas, determinar costos y facturar a los usuarios. CAFETO no provee, por no ser de interés para esta tesis, mecanismos para ejecutar *accounting* de ningún tipo.

## P de Performance

***Performance***, involucra analizar o medir el rendimiento de la red en función de métricas. Algunos ejemplos de métricas de performance son:

- *Rendimiento*, medido por la cantidad de unidades de comunicación durante un período de tiempo. Dependiendo de la capa que se mide, se usan diferentes unidades, por ejemplo:

- En la capa de enlace, el número de *bytes* u octetos, que se transmiten por segundo.
- En la capa de red, el número de paquetes que se envían por segundo.
- En la capa de aplicación para un servicio web, el número de peticiones web que atiende un servicio por segundo.
- En la capa de aplicación para un servicio de voz, el número de llamadas de voz que pueden ser procesados por hora.

- *Demora*, medida en unidades de tiempo. Pueden medirse diferentes tipos de demora, dependiendo de la capa con la que se está tratando. Algunos ejemplos son los siguientes:

- En la capa de enlace, el tiempo que tarde un octeto en llegar a su destino en el otro extremo de la línea.
- En la capa de red, el tiempo que tarda un paquete IP para llegar a su destino.
- En la capa de aplicación para un servicio de voz, el tiempo que tarda en recibir un tono de llamada después de haber levantado el receptor.

La *performance* se ocupa del monitoreo de la red para mejorar o mantener su rendimiento. Incluye una amplia variedad de funciones como la visualización del estado actual de la red, donde se observa el valor de algunos parámetros de performance o parámetros en el tiempo, a través de la traza de un histograma. Los histogramas son resultado de la observación de estos parámetros cada una determinada cantidad de segundos durante un período tiempo. Estos gráficos permiten por ejemplo, distinguir entre una caída repentina y un aumento en el valor de un atributo que en general se mantiene entre otros valores.

Algunos patrones pueden indicar que un problema está a punto de ocurrir, por ejemplo, un aumento de la utilización de una interface puede preceder a un aumento en el número de paquetes que se descartan, que, a su vez, puede preceder a que los usuarios comiencen a tener mayor tiempo de espera en sus aplicaciones. La administración de la performance permite anticipar los problemas y ocuparse de ellos antes de que ocurran.

La observación de valores de atributos críticos durante un período de tiempo permite descubrir tendencias. Si por ejemplo crece continuamente la utilización de un enlace o se detectan cuellos de botella –áreas constantemente congestionadas- se podría planificar una actualización o por el contrario, si se detectan áreas subutilizadas se podría pensar cómo aprovechar mejor ese equipamiento. Toda esta información puede ser muy valiosa para ajustar la configuración de la red y lograr una óptima performance.

Desde la consola de CAFETO se pueden invocar consultas predefinidas o generar nuevas consultas combinando distintos atributos que pertenecen a los grupos de las MIBs. Cada una de estas consultas pueden ser guardadas y ejecutadas posteriormente. La funcionalidad del agente podría extenderse implementando nuevas consultas para medir performance basadas en fórmulas que hacen uso de los distintos

atributos de los grupos de las MIBs, por ejemplo tasa de paquetes recibidos/enviados, tasa de errores, tasa de utilización, que puede ser por interface o por protocolo, etc.

Los resultados provenientes de las consultas son registrados en una base de datos para ser mostrados posteriormente mediante reportes textuales o gráficos estadísticos que describen visualmente los datos analizados. Estos gráficos podrían ser diagramas de tortas, barras, histogramas, etc. Existen APIs como *JFreeChart*[1] para la plataforma JAVA que permiten la generación de estos diagramas.

## S de Security

La letra final en FCAPS, la S, fue incluida para el área de seguridad. El gerenciamiento de seguridad es el proceso que involucra el control de acceso a la información y a los datos de la red. Incluye aspectos como la prevención de ataques provenientes de hackers o propagación de virus o la detección de intentos de intrusión maliciosa.

Dos aspectos deben ser distinguidos: la ***seguridad del gerenciamiento***, que asegura que el propio gerenciamiento sea seguro, y el ***gerenciamiento de la seguridad***, que involucra administrar la seguridad de la red.

### *Seguridad del gerenciamiento*

La seguridad del gerenciamiento está relacionada con asegurar que las operaciones de gerenciamiento sean ellas mismas seguras. Para ello el acceso a la aplicación y también el acceso a las operaciones de gestión deben estar restringidos a usuarios autorizados. No sólo se debe prevenir de accesos indebidos desde afuera de la red –definiendo usuarios y contraseñas para el acceso al sistema- sino también se deben prevenir accesos no autorizados desde adentro de la red definiendo por ejemplo, privilegios de accesos a las operaciones de gerenciamiento.

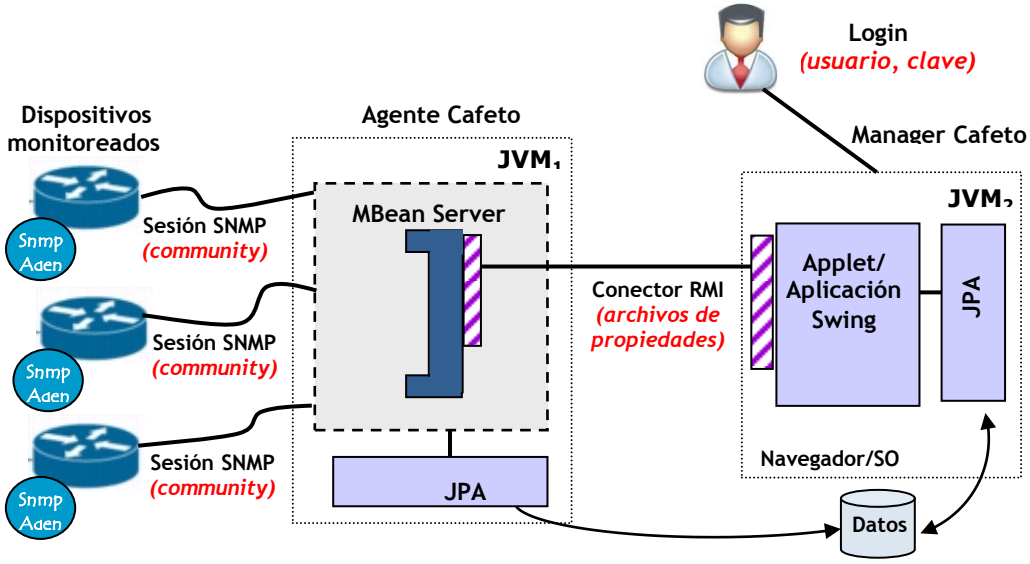



**Figura 6.5: Seguridad del Gerenciamiento**

[1] Sitio web del proyecto JFree: http://www.jfree.org/jfreechart/-.

Cafeto, como se describe en el item 6.4.2, establece que para acceder a la aplicación de gerenciamiento se requiere de una cuenta de usuario que pertenezca a un perfil de usuario. Para este prototipo se han definido dos perfiles: administradores y monitores, los cuales tienen diferentes permisos de acceso.

Una vez ingresado a la aplicación manager, para comunicarse con los agentes JAVA se utilizan socket seguros basados en Secure Sockets Layer o SSL que hacen uso de archivos de propiedades como indica la Figura 6.5. Los archivos de propiedades contienen las contraseñas, roles y niveles de acceso. Estos archivos se utilizan para autenticar al cliente y para autorizarlo; las operaciones de gerenciamiento se llevarán a cabo sólo si la autenticación fue exitosa. Uno de los archivos **`password.properties`**, se usa para autenticar a los usuarios, define roles y contraseñas para los diferentes roles. El otro archivo de propiedades **`access.properties`**, se utiliza para controlar los accesos remotos a los recursos del agente a través de la API JMX. Este archivo define los accesos permitidos para los diferentes roles. Una entrada consiste en el nombre de un rol y el nivel acceso asociado. Fragmentos de código java con detalles de la implementación de estos aspectos de seguridad están disponibles en el Anexo A, páginas 93-95.

http://www.ibm.com/developerworks/ssa/java/library/j-mxbeans/#resources (SSL)

En cuanto a la seguridad del propio protocolo SNMP, Cafeto soporta autenticación basada en *communities* para requerimiento SNMPv1 y SNMPv2c. Cuando un agente recibe un requerimiento SNMPv1 o SNMPv2 proveniente de un manager, chequea por autenticación con el *community string* recibido y el tipo de requerimiento SNMP (GetRequest, GetNextRequest, SetRequest, etc.).

Además de esta autenticación para requerimientos v1/v2, el agente podría soportar acceso basado en vistas cuando el VACM está habilitado. Como ya se ha mencionado en el capítulo 3 donde se describe SNMP, el VACM permite restringir el acceso a sólo un subconjunto de la información de gestión, es decir, se podrían definir vistas de las MIBs -subconjunto de la información de gerenciamiento- y permitir determinados accesos solo para ese subconjunto. La API provee soporte para manejo de seguridad SNMPv3.

### *Gerenciamiento de la seguridad de la red*

El gerenciamiento de la seguridad involucra administrar la seguridad de la red en sí misma, como opuesto a la seguridad usada para el gerenciamiento. Este gerenciamiento le permite a los administradores limitar el acceso a los recursos y dispositivos de la red para proteger la información sensible.

Algunas amenazas comunes de seguridad incluyen: ataques de hackers para tratar de obtener el control de un sistema, propagación de virus, etc. Para protegerse de esas y de otras amenazas de seguridad se podrían utilizar deferentes herramientas como software para monitoreo el tráfico de la red con el objetivo de para detectar tráfico sospechoso que podría indicar un ataque en curso o se podrían aplicar políticas para limitar o permiten solamente un aumento gradual de tráfico hacia un destino particular o que se origine de una fuente particular, así si un ataque da lugar a una repentina ráfaga de tráfico, esta técnica permite una degradación más lenta de la red y sus servicios si ella está siendo atacada.

Una herramienta visual para gerenciamiento también podría mostrar en el mapa de red los lugares donde se han definido medidas de seguridad. Un administrador podría seleccionar cualquier dispositivo de la red para analizar o cambiar las políticas de

seguridad establecidas para dicho dispositivo. La aplicación también podría monitorear los intentos de acceso a puntos con información sensible, informar cualquier anomalía por ejemplo cambiando el color del dispositivo en cuestión o creando reportes que describan accesos denegados reiterados a un dispositivo. Estas funcionalidades podrían incorporarse a Cafeto.

## 6.3 Descripción del monitoreo de Cafeto

De las secciones anteriores se desprende que de las áreas de FCAPS, CAFETO cubre especialmente las relacionadas con manejo de *fallas* y *performance,* en esta sección explicaré cómo se lleva a cabo el monitoreo de datos para cubrir estas áreas.

Los servicios de monitoreo -para administrar fallas y medir la performance- provistos por CAFETO son configurados y activados desde la consola de gerenciamiento pero es el agente CAFETO quien los administra. Estos servicios son representados –en el agente- según la especificación JMX como *MBeans*.

La **Figura 6.6**, recuerda la arquitectura del agente JAVA donde pueden observarse el MBeanServer con dos MBeans registrados, `StringMonitor` y `CounterMonitor`. Estos MBeans son monitores que permiten comparar varios datos y enviar notificaciones al manager bajo circunstancias específicas.

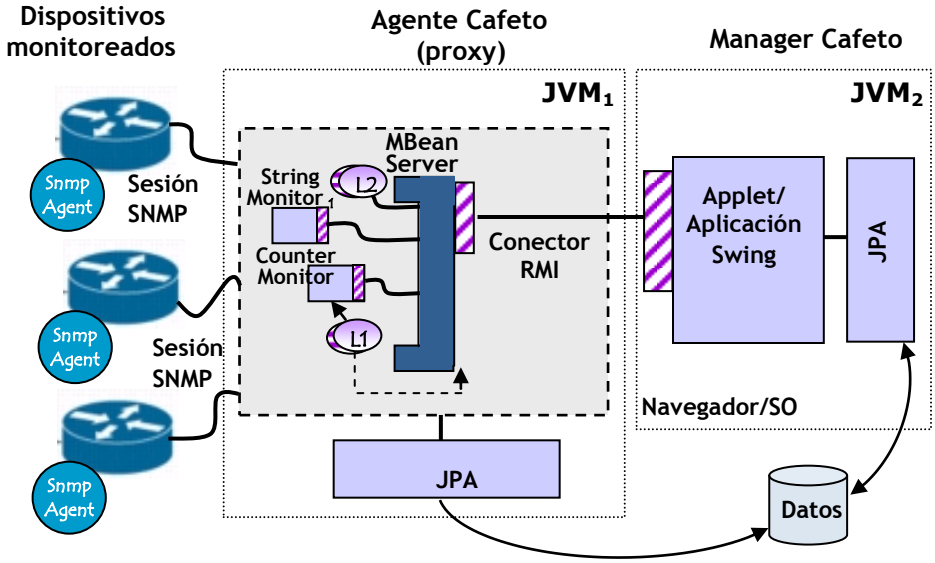



**Figura 6.6: Arquitectura del agente CAFETO**

Los atributos de los *MBeans* del agente son monitoreados usando los servicios de monitoreo de JMX [JSR 255].

*¿Cómo monitorea el agente CAFETO los atributos del sus MBeans?*

Los servicios de monitoreo del agente CAFETO permiten observar la variación en el tiempo de los valores de los atributos en los MBeans y emitir notificaciones ante determinadas situaciones. Los monitores envían notificaciones cuando los valores observados reúnen ciertas condiciones, principalmente cuando igualan o exceden un *threshold*. Las condiciones pueden ser especificadas cuando el monitor es inicializado o dinámicamente desde la aplicación de gerenciamiento CAFETO usando la interface del respectivo Monitor –que también es un MBean-.

CAFETO provee tres tipos de monitores para obtener valores de un atributo:

- *CounterMonitor:* observa atributos con tipos de datos enteros java (Byte, Integer, Short, Long) que se comportan como contadores.
- *GaugeMonitor*: observa atributos con tipos de datos java enteros o punto flotante (Float, Double) que se comportan como un *gauge* (incrementan o decrementan arbitrariamente).
- *StringMonitor:* observa un atributo de tipo String.

Todos los monitores tienen un período de granularidad configurable, que determina cada cuanto tiempo se consultará -*pulled*- el valor del atributo.

Se denomina *derived gauged* al valor observado por el monitor, que puede ser el valor exacto obtenido de un atributo en un momento determinado o la diferencia de valor de dos observaciones consecutivas. El intervalo de tiempo durante el cual un atributo es monitoreado es llamado *período de granularidad* y es especificado en milisegundos. También se puede configurar cuanto se debe incrementa el valor del *threshold* una vez que alcanzó el valor tope –este valor es conocido como *offset*-.

Los monitores emiten notificaciones cuando se alcanza una condición -por ejemplo, se alcanza un *threshold*- o cuando se detecta un error -por ejemplo, se desea observar un atributo de un MBean que no está registrado-. Todos los tipos de monitores emiten esta notificación usando un objeto de tipo **`MonitorNotification`**. De este objeto **`Notification`** podemos obtener el tipo de evento que se ha producido, el nombre del MBean monitoreado, el nombre del atributo observado, el valor observado y el valor umbral o *string* con el que se comparó.

Todos los monitores envían notificaciones de error y otras notificaciones de acuerdo al tipo de monitor. La **Figura 6.7** ilustra todos los tipos de notificación definidos en la API JMX. El agente instancia y envía a la aplicación de gerenciamiento estas notificaciones.

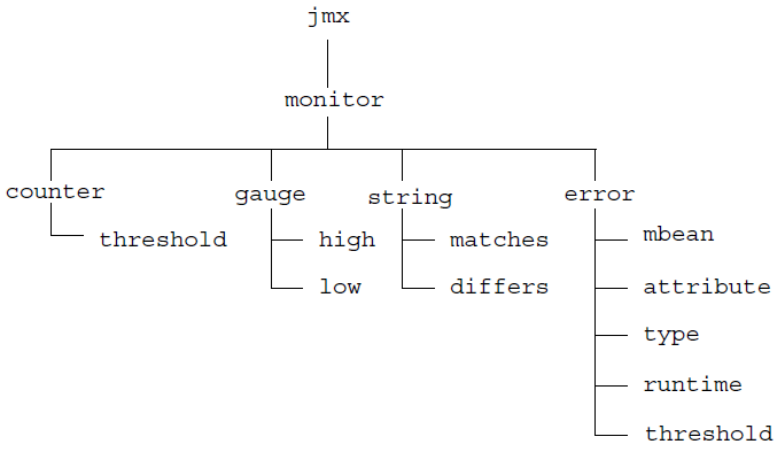



**Figura 6.7: Tipos de notificaciones de los monitores**

### 6.3.1 Monitor de contadores

Un **CounterMonitor** es un tipo de monitor que envía una notificación cuando el valor del contador observado alcanza o excede un nivel de comparación, conocido como *threshold*. La **Figura 6.8** ilustra el funcionamiento de este tipo de monitor usando offset.

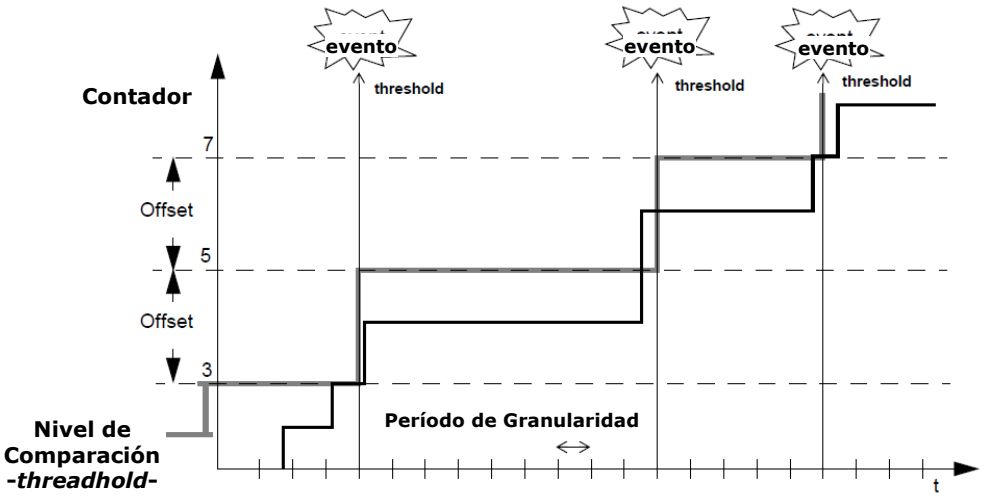



**Figura 6.8: Funcionamiento de un *CounterMonitor* con *offset***

El monitor observa un contador que varía en el tiempo t. Después de que un *threshold* es alcanzado, ese umbral es incrementado con un valor –*offset*- hasta que el nivel de comparación o *threshold* es mayor que el valor actual del contador o valor observado.

Además de las notificaciones de errores, un objeto de tipo **`CounterMonitor`** puede enviar notificaciones cuando se alcanza o excede un valor umbral, de manera que estos monitores pueden usarse para manejar fallas o medir performance.

Para crear este tipo de monitor con el Manager de CAFETO, se debe seleccionar el recurso sobre el que se quiere registrar el monitor y definir los parámetros necesarios para configurar el monitor. La **Figura 6.9** muestra la ventana de diálogo que se abre cuando se quiere definir un nuevo monitor.

Los campos que se deben ingresar para generar un nuevo *CounterMonitor* son los siguientes:

- *Nombre*: un nombre que identifica junto con una dirección IP al monitor que se está creando.
- *Atributo*: se debe seleccionar un grupo de MIB en la lista de selección que se encuentra a la izquierda y luego un atributo de ese grupo usando la lista de la derecha. En esta lista aparecen únicamente los atributos de la MIB II, que son de tipo `Integer`, `Timeticks`, `Integer32`, `Counter`, `Counter32`, `Counter64` porque este monitor trabaja con contadores.
- *Threshold*: se debe especificar el valor de comparación, para enviar una notificación cuando el valor observado lo supere
- *Offset*: Especifica el valor que se le debe incrementar al *threshold* después de que el valor observado alcanzó o superó dicho valor.
- *Periodicidad* (período de granularidad): Especifica el intervalo de tiempo entre los monitoreos u observaciones del valor del atributo.
- *Tipos de Notificación*: Provee la configuración de mecanismos de notificación, para informar a los administradores la ocurrencia de un evento importante.

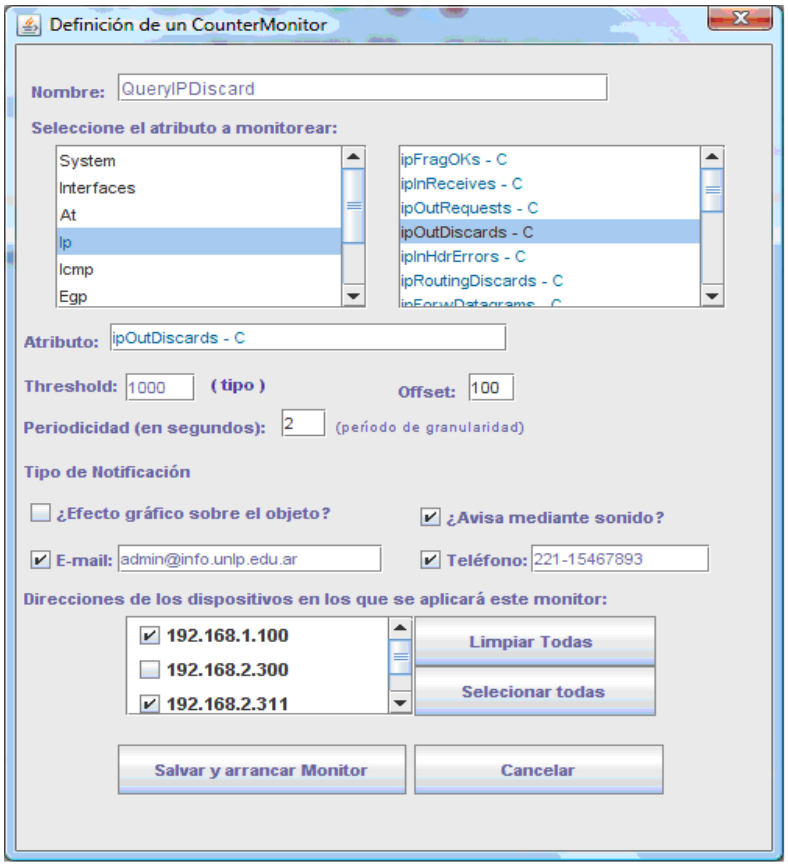



**Figura 6.9: Definición remota de un CounterMonitor**

**Ejemplo**: Supongamos que queremos definir un *CounterMonitor* para monitorear el atributo **`ipOutDiscards`** definido en la MIB II donde el nivel de comparación o *threshold* sea 1000, el valor de offset 100 y el período de granularidad 2000 -el atributo deberá ser monitoreado cada 2 segundos-.

Si se elige algún tipo de notificación, el monitor enviará la notificación cuando el valor del atributo **`ipOutDiscards`** alcance el valor 1000. El valor de comparación 1000 será incrementado en 100 -offset-, luego de que se alcance el nivel de comparación.

La Figura 6.8, ilustra la ventana para configurar un *CounterMonitor* la cual dispone de un botón "Salvar y Arrancar Monitor" para crear y arrancar un nuevo monitor desde la consola remota de gerenciamiento.

### 6.3.2 Monitor de Gauges

Un ***GaugeMonitor*** es un monitor que se utiliza para observar a un atributo de tipo entero o punto flotante que fluctúa dentro de un rango. Este monitor tiene un *threshold* superior –**`highThreshold`**- y uno inferior –**`lowThreshold`**-, cada uno de los cuales puede disparar diferentes notificaciones. Las notificaciones serán emitidas solamente si fueron configurados a true los atributos **`NotifyHigh`** y **`NotifyLow`** del monitor.

Inicialmente si **`NotifyHigh`** fue seteado a *true* y el valor observado alcanza el valor del **`highThreshold`**, entonces se dispara una notificación. Luego, a pesar de que se vuelva

a superar el **highThreshold**, no se notifica si antes no se alcanzó un valor más bajo que **lowThreshold**. De similar manera se comporta cuando **NotifyLow** es seteado a *true*, notifica cuando se alcanza un valor inferior a éste y no vuelve a notificar si antes no se supera el valor de **highThreshold**. Además de las notificaciones de errores, un monitor de este tipo puede enviar notificaciones especiales cuando el gauge derivado excede el valor del atributo **thresholdHigthValue** o cuando es inferior a **thresholdLowValue**.

La **Figura 6.10** ilustra el funcionamiento de un monitor de este tipo.

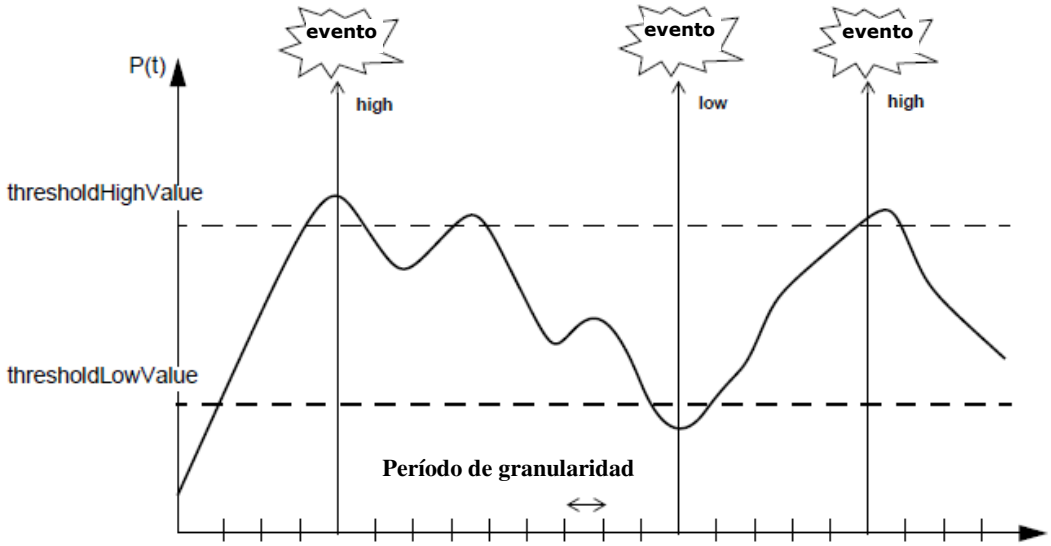



**Figura 6.10: Funcionamiento de un *GaugeMonitor***

De manera similar al **CounterMonitor**, para crear este tipo de monitor se deben ingresar los siguientes campos:

- *Nombre*: un nombre que identifica junto con una dirección IP el monitor que se está creando.
- *Atributo*: se debe seleccionar uno de los atributos de la lista que figura a la derecha. En esta lista aparecen únicamente los atributos de la MIB II, que son de tipo Gauge32 y Unsigned32.
- *Periodicidad* (período de granularidad): Especifica el intervalo de tiempo entre las observaciones del valor del atributo.
- *Tipos de Notificación:* Provee la configuración de mecanismos de notificación, para informar a los administradores la ocurrencia de un evento importante. Además se puede configurar si la notificación es por el *threshold* superior, inferior o por ambos.

### 6.3.3 Monitor de Strings

Un ***StringMonitor*** es un monitor que observa atributos de tipo *String*. Este monitor ejecuta una comparación exacta entre un *String* determinado y el valor observado en un atributo de tipo *String* de la MIB. Un monitor de este tipo, envía una notificación

cuando el valor del atributo es igual o cuando es distinto a un *String* determinado, según se lo haya configurado.

Para que el monitor emita una notificación cuando un atributo observado es igual a un determinado String, se debe setear el atributo NotifyMatch a *true*. De manera análoga, para que emita una notificación cuando un atributo observado es distinto a un determinado *String*, se debe setear el atributo NotifyDiffer a true. Si se seleccionan ambas notificaciones, el monitor avisa solo cuando los valores van cambiando alternativamente. La **Figura 6.11** ilustra el funcionamiento de un StringMonitor.

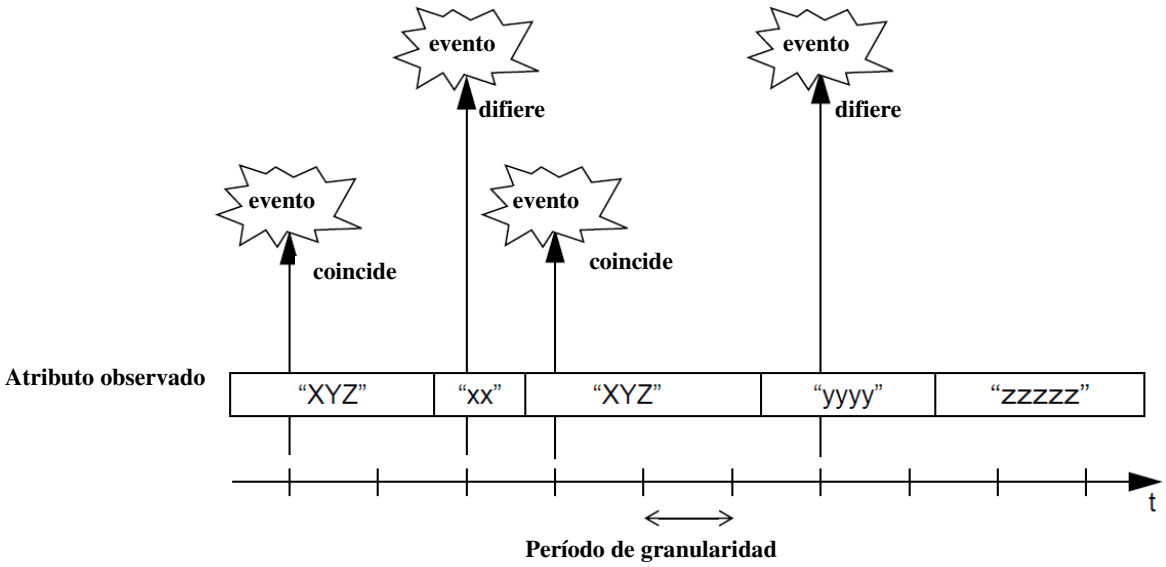



**Figura 6.11: Funcionamiento del `StringMonitor`**

Además de las notificaciones de errores, un StringMonitor puede enviar notificaciones cuando el *String* observado es igual o difiere del contenido del String con el que se está comparando. De manera similar a los anteriores monitores, para crear este tipo de monitor se deben ingresar los siguientes campos:

- *Nombre*: un nombre que identifica junto con una dirección IP el monitor que se está creando.
- *Atributo*: se debe seleccionar uno de los atributos de la lista que figura a la derecha. En esta lista aparecen únicamente los atributos de la MIB II, que son de tipo String.
- *String a comparar*: se debe entrar una cadena de caracteres o String que se comparará con el valor del atributo.
- *Periodicidad* (período de granularidad): Especifica el intervalo de tiempo entre las observaciones del valor del atributo.
- *Tipos de Notificación*: Provee la configuración de mecanismos de notificación, para informar a los administradores que el String que se está comparando ha cambiado y es igual o diferente al del atributo de la MIB.

*¿Cómo es el monitoreo de varios agentes SNMP?*

Como ya se analizó en el capítulo anterior, cada agente CAFETO puede monitorear a uno o más agentes SNMP. Cada agente JAVA puede ejecutar en la misma máquina donde ejecuta el agente SNMP o en otra máquina con acceso para monitorear a uno o más agentes SNMP ubicados en otros dispositivos. La información sobre los agentes SNMP que debe monitorear el agente JAVA se definen en un archivo de propiedades que lee el agente JAVA en la etapa de arranque.

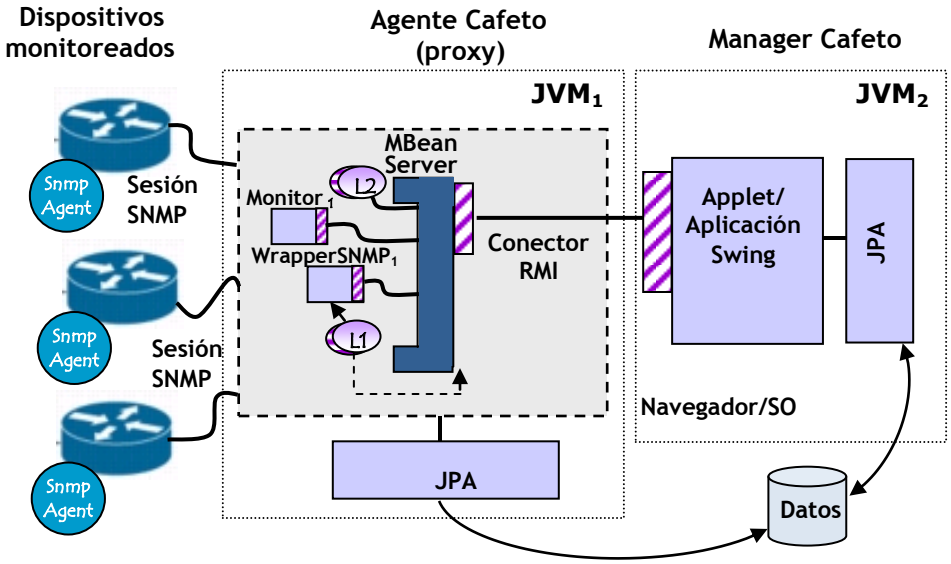



**Figura 6.12: Comunicación entre el Agente JAVA y los agentes SNMP**

Cuando desde una aplicación de gerenciamiento se quiere definir por ejemplo un nuevo monitor, se tiene la posibilidad de activarlo para todos los agentes SNMP que gerencia o sólo para algunos. En la **Figura 6.9** se puede observar que cuando se configura remotamente un nuevo monitor para un agente JAVA se despliega una lista con las direcciones IP de los dispositivos donde están corriendo los agentes SNMP de manera que seleccionando el *check-box* se activará el monitor para ese agente SNMP.

## 6.4 Otras características de CAFETO

En las secciones anteriores se ha descripto a CAFETO desde el punto de vista funcional mostrando algunas pantallas de la interfaz de usuario. En esta sección se describen otros aspectos que se tuvieron en cuenta para la definición de la consola de la aplicación de gerenciamiento y la definición de los roles para restringir el acceso a determinadas funcionalidades y recursos.

### 6.4.1 Características de CAFETO desde el punto de vista de la Interfaz de Usuario

Para la definición de la interfaz de usuario de CAFETO, se ha estudiado el paradigma de interfaces de usuarios colaborativas y se lo aplicó al gerenciamiento de redes de datos. La incorporación de aspectos colaborativos en la interfaz de usuario, fue esencialmente motivada por la complejidad que existe en el gerenciamiento de las redes de datos, y creo que la cooperación en la administración de los recursos, le

permite a los usuarios del sistema unir sus experiencias y conocimientos para lograr un eficiente gerenciamiento de la red.

Las interfaces colaborativas están evolucionando desde el modelo estrictamente WYSIWIS [Stefik, M.] en el cual todos los participantes ven exactamente las mismas cosas en todo momento, hacia vistas en donde los usuarios podrían ver diferentes partes de los espacios compartidos [Baecker, R.].

Los usuarios usando interfaces WYSIWIS más relajadas, necesitan conocer acerca de las interacciones de los otros usuarios en el espacio compartido, esta capacidad de ver y conocer sobre las actividades de sus pares, es conocida en la bibliografía como *awareness*. La tabla de la **Figura 6.** ilustra el uso de *awareness* en la interfaz de usuariop aplicación de gerenciamiento.

| ¿Qué awareness soporta? | ¿Cómo lo hace? |
|---|---|
| **¿Quiénes están conectados a CAFETO?** | La interfaz de CAFETO provee un sector de la pantalla con cada uno de los nombres de los usuarios conectados identificados por su imagen o ícono elegido. |
| **¿Dónde están trabajando los otros usuarios de CAFETO?** | La interfaz muestra dónde están trabajando los otros usuarios situando la cara de ellos al lado del elemento gerenciado sobre el que está trabajando. |
| **¿Qué están haciendo los otros usuarios de CAFETO?** | Existe una lista de usuarios conectados a CAFETO. Si se selecciona a uno de ellos se puede comenzar una conversación y conocer –según su rol- las actividades que están desarrollando. |

**Figura 6.: Tabla de *awareness* soportados por CAFETO**

### 6.4.2 Usuarios de CAFETO

CAFETO clasifica a los usuarios en *administradores* con permiso para utilizar cualquier funcionalidad del sistema y *monitores* con la posibilidad de ver el estado de la red, monitorear, visualizar aspectos de performance, pero sin atribuciones de configuración.

| Tipos de Usuario | | Administrador | Monitor |
|---|---|---|---|
| **Funciones de Gerenciamiento** | Configuración | Agregar/Eliminar elementos gerenciados. Ver/Setear atributos de elementos gerenciados. | Ver atributos de Elementos gerenciados. |

| | | | |
|---|---|---|---|
| | Performance | Pedir/Modificar/Borrar consultas sobre elementos gerenciados. | Pedir/Borrar sus consultas sobre elementos gerenciados. |
| | Manejo de Fallas | Agregar/Eliminar la activación de alarmas.<br>Configurar distintas formas de recepción de alarmas. | No pueden manejar alarmas. |
| **Funciones de Colaboración** | Libro de Fallas | Ver/Actualizar/Borrar información del Libro. | Ver/Actualizar información del Libro. |
| | Mensajería | Enviar y recibir mensajes | Enviar y recibir mensajes |
| | Chats | Iniciar nuevos chats con usuarios de CAFETO. | Iniciar nuevos chats con usuarios de CAFETO. |
| | *Awarness* de quiénes están | SI | SI |
| | *Awarness* de qué están haciendo | SI | Sólo de chats activos |
| | *Awarness* de dónde están | SI | SI |
| **Ambito de Gerenciamiento** | Acceso a la Red | Toda la Red de datos | Sólo de determinados elementos de gerenciamiento |

**Figura 6.: Funcionalidades permitidas por tipo de usuario**

Los *administradores* son los encargados de crear nuevos usuarios. Ellos tienen privilegios para crear nuevos *administradores* y nuevos *monitores*. Los usuarios encuadrados como *monitores*, si bien comparten las mismos privilegios, pueden diferenciarse por el ámbito de gerenciamiento que tienen asignado. Un *administrador* podrá definir, por ejemplo, que un *monitor* administre una cierta red LAN, o que tenga acceso a ciertos Routers, o que sea el encargado de monitorear todas las impresoras de la red.

Otro aspecto importante de los perfiles de usuario, es que cada categoría tendrá una interfaz de usuario en la que podrá operar sólo con funciones y objetos habilitados. Esto significa que todos los usuarios tienen una misma visualización lógica de la red, los mismos íconos, los mismos menúes y funcionalidades aparentes, pero únicamente podrán utilizar aquellos objetos habilitados.

Estos aspectos de colaboración de la interfaz de usuario y de los privilegios de cada perfil no fueron implementados en este prototipo. En el próximo ítems, analizaré las funcionalidades implementadas desde el punto de vista del gerenciamiento y del uso de la API JMX para monitoreo de la red usando SNMP.

## Síntesis

En este capítulo he analizado el modelo FCAPS, detallando cada una de las áreas funcionales y mostrando cómo el prototipo correspondiente a la aplicación de gerenciamiento y al agente JAVA cubren esas áreas.

El *manejo de fallas* consiste en funciones para vigilar la red para asegurar que todo está
funciona correctamente. El manejo de alarmas y de los eventos que están siendo constantemente generados es uno de los desafíos de la gestión de fallos.

El *gerenciamiento de la configuración* se refiere a cómo es configurada la red. Esto involucra la configuración de parámetros de manera que la red pueda proporcionar los servicios que se espera. La configuración también incluye funciones que permiten descubrir lo que hay en ella.

El *manejo de accounting* está relacionado con la recolección y registro de datos acerca de la utilización de la red y el consumo de sus servicios por parte de los usuarios finales.

La *administración de performance* tiene que ver con la recopilación de estadísticas de la red para evaluar su desempeño y ajustarla. El objetivo es permitir la correcta asignación de los recursos en la red -como la eliminación de cuellos de botella- para poder brindar la mejor calidad de servicio posible con los medios disponibles.

Por último, la *gestión de la seguridad* está dirigida a prevenir diversos tipos de amenazas a la seguridad que una red.

# Capítulo 7

## Conclusiones y Trabajos Futuros

La proliferación de las redes en todo tipo de organizaciones, en conjunción con la distribución geográfica de sus dependencias, genera un escenario donde los dispositivos de red, servidores y servicios no se encuentran concentrados en una sola ubicación. Si a esto le sumamos la importancia de tener disponibilidad en los servicios 7X24, muchas veces conduce a que los administradores utilicen diversas técnicas y herramientas de monitoreo para brindar respuestas inmediatas ante potenciales anomalías, tanto en los servicios como en la infraestructura en la que se montan los mismos.

Es cierto que muchos fabricantes brindan al usuario herramientas para el monitoreo de sus equipos pero, gran parte de las mismas utilizan protocolos propietarios lo que impide la integración y la correlación con la información generada por los equipos de otros fabricantes. Por otro lado, también existen herramientas que se apoyan en protocolos estándares como SNMP para el gerenciamiento de los dispositivos remotos, que facilitan el monitoreo de recursos, aunque muchas de ellas generan mucho tráfico SNMP, necesitan acceder a la estructura interna de la red y otras, además poseen aplicaciones de gerenciamiento con interfaces de usuario muy textuales y poco amigables.

Este escenario, motivó el estudio de SNMP: funcionamiento, versiones, limitaciones y también el análisis de las APIs disponibles relacionadas con este protocolo con el objetivo de proponer alguna herramienta que ayude a los administradores de red en su labor cotidiana. A continuación se describen algunas conclusiones sobre la arquitectura propuesta y sobre los aspectos abordados para su implementación.

### 7.1 Sobre la Arquitectura propuesta

En este trabajo se ha propuesto una arquitectura para una aplicación de gerenciamiento y para un agente, que facilite el monitoreo de redes de dispositivos con SNMP y se la ha implementado a través de un prototipo JAVA.

En un principio sólo se pensó en desarrollar un agente JAVA que funcione como un wrapper de un agente SNMP de manera de extender las funcionalidades provistas por el mismo. Esta solución podría ser viable cuando el número de agentes SNMP es pequeño pero, cuando la cantidad es muy grande, destinar un agente JAVA distinto para manejar cada uno de ellos, resultó complejo y a veces inviable -por ejemplo cuando el dispositivo a monitorear no soporta java o se encuentra inaccesible por aspectos de seguridad-. Para disminuir esta complejidad, sin interferir en las políticas de seguridad de la red a monitorear, se propuso una arquitectura donde un agente

java o *master* concentre y envíe a la aplicación de gerenciamiento toda la información de monitoreo correspondiente a un conjunto de agentes SNMP.

Esta propuesta no sólo simplifica la administración de los dispositivos sino también permite:

- Reducir la carga de la red, dado que en vez de que la aplicación de gerenciamiento hable con cada agente SNMP, el propio agente JAVA lo hace. De esta manera el agente JAVA envía notificaciones a la aplicación únicamente cuando es necesario. Cabe aclarar que las notificaciones no son simplemente TRAPs SNMP sino mensajes disparados por el agente JAVA ante deteminadas situaciones.
- Aprovechar la masiva disponibilidad de SNMP, y a su vez, extender la funcionalidad provista por el mismo. No se puede extender la funcionalidad de un agente SNMP, este agente sólo puede exponer la información que está en la MIB; la funcionalidad de un agente JMX no tiene límite, se pueden definir las funcionalidades que se desee como se demuestra en esta tesis.
- Darle a los agentes JAVA, inteligencia y autonomía, características carentes en los agentes SNMP. El agente puede, entre otras cosas, monitorear dispositivos, determinar si ciertos valores alcanzan un límite y reportar toda esta información a la aplicación de gerenciamiento, a un administrador o realizar alguna acción. Estos son ejemplos de funcionalidades incorporadas en el agente pero se podría implementar cualquier otra funcionalidad. Cabe destacar que los agentes SNMP pueden avisar sobre algún cambio de valor en un atributo, pero no pueden por ejemplo correlacionar eventos, ni disponer de algún tipo de lógica compleja.
- Incrementar la funcionalidad del agente (JAVA) en *run-time* agregando desde el manager, nuevas MIBs y servicios al agente, sin detenerlo.
- Persistir los datos de gerenciamiento en un base de datos. El agente del prototipo incorpora nuevos servicios como la persistencia de la información de gerenciamiento en base de datos, proveyendo una importante fuentes de información para análisis, correlación de eventos, detección de tendencias, etc. Asimismo esta estos datos podrían ser usados para graficar histogramas o cualquier tipo de diagrama.

## 7.2 Sobre la API JMX

Después de haber utilizado la API para gerenciamiento de JAVA, puedo concluir que la misma brinda un framework muy general, no define monitoreos específicos, ni estructuras de datos de gerenciamiento. Queda a cargo de los desarrolladores establecer los mecanismos para extraer la información de los objetos gerenciados para ser analizados. Sin embargo, JMX define una API que ayuda en la definición de agentes y para la comunicación entre los agentes y gerentes.

En cuanto a la instrumentación de los recursos o servicios, puedo decir que la API ofrece varios tipos de MBeans. Para la mayoría de los casos los MBeans estándares, que son los más simples de instrumentar, son suficientes; sin embargo existen otros tipos de MBean especiales para modelar recursos dinámicos. El tipo de MBean sólo determina cómo será desarrollado el MBean pero no cómo será manejado, el agente (MBean Server) maneja a todos los tipos de MBean en forma transparente.

En cuanto a los servicios de monitoreo provistos por la API, uno de los aspectos más analizados para este trabajo, existen básicamente dos opciones para hacerlo: crear un *listener*, objeto que escucha por determinados cambios en el Mbean o definir un *monitor* que periódicamente consulte al MBean y emita notificaciones solamente cuando existe un evento que así lo amerite. *¿Qué mecanismo recomiendo?* Dependiendo de la complejidad de la situación se podría elegir una u otra opción. Si los requerimientos son simples, recomiendo registrar un listener directamente en el

MBean porque un MBean le envía su notificación a su listener y éste envía dicha notificación inmediatamente al manager. La única limitación de este mecanismo -según la especificación J2SE 5.0, paquete javax.management- es que el listener no provee facilidades para comparar valores con *threadholds*. Por este motivo, cuando se necesita un mecanimso más complejo para notificar o si se quieren monitorear y correlacionar varios cambios en los valores de atributos de un MBean, recomiendo definir un Monitor. Los monitores, especificados en la J2SE 5.0, son MBeans que permiten comparar varios datos y enviar notificaciones bajo circunstancias específicas. Sin embargo, con este mecanismo, el monitor debe consultar periódicamente a los MBeans para detectar cambios en los valores de sus atributos y notificar cambios a los listeners interesados.

## 7.3 Sobre SNMP y JMX

Como ya se ha descripto en el capítulo 3, el SNMP se basa en la idea de que la gestión de la información se organiza en MIBs, con variables de gerenciamiento individuales -u objetos gerenciados-, identificados a través de identificadores de objetos (OID). SNMP proporciona un pequeño conjunto de primitivas que permiten a un administrador leer y escribir valores en las MIBs, y un agente para enviar eventos a las aplicaciones de gerenciamiento. SNMP tienen tres versiones, cada una superadora de su antecesora, sin embargo, hoy en día se puede encontrar dispositivos usando cualquiera de ellas.

*¿Por qué trabajé con SNMP?* El protocolo SNMP es quizá el protocolo de gerenciamiento más conocido. También es el protocolo de gerenciamiento que más se ha adoptado para aplicaciones de monitoreo, está disponible en la mayoría de los dispositivos y es además, el protocolo de gerenciamiento que nos han enseñado en uno de los cursos de la maestría.

El protocolo SNMP es el más popular para gerenciamiento de redes pero tiene algunas limitaciones. El poder de expresión de SNMP/SMI es limitado. Se ajusta bien para describir y monitorear valores numéricos (*gauges*), contadores (escalares y tabulares) y *strings*, pero para describir datos complejos es mucho más difícil. SNMP es un muy buen protocolo pero la posibilidad de evolución de las interfaces SNMP es limitada, como cualquier tecnología de gerenciamiento tiene fortalezas, debilidades y limitaciones.

*¿Cuál es el aporte o beneficio del agente JAVA respecto del agente SNMP?* El agente Cafeto es como un wrapper de un agente SNMP, de este modo extiende sus funcionalidades. Como es sabido, el protocolo SNMP es un protocolo relativamente simple, con funcionalidades limitadas, pero el uso de JAVA hace que las funcionalidades del agente no tengan límite. Además, los agentes SNMP pueden ser monitoreados por managers SNMP -como HP OpenView, IBM Tivoli, etc.-, pero los agentes JMX además de poder ser monitoreados por consolas SNMP pueden ser administradas por aplicaciones JAVA.

## 7.4 Tendencias actuales y trabajos futuros

La API JMX ha tenido buena aceptación en servidores de aplicaciones JEE muy populares como BEA WebLogic y JBoss, los cuales han adoptado esta tecnología para administrar y monitorear el estado de sus componentes. Otro uso puede encontrarse en la propia distribución de la pltaforma estándar de JAVA, la cual provee una consola para monitorear el estado de la propia máquina virtual JAVA.

Finalmente, creo que SNMP and JMX se complementan muy bien para facilitar la implementación de aplicaciones de gerenciamiento basadas en este estándar. SNMP es un protocolo estable, ampliamente utilizado, que ha evolucionado mucho y JMX es una tecnología nueva que promete seguir creciendo.

En cuanto a los trabajos futuros vinculados con esta tesis hay dos líneas que podrían seguirse: una está relacionada con el desarrollo de agentes JMX sin intervención del protocolo SNMP y la otra línea podría estar vinculada con profundizar el estudio de otras categorías de FCAPS como seguridad que no ha sido abordada.

# Anexo

## Segmentos de Código

La implementación de este prototipo involucró varias capas de software. La aplicación de gerenciamiento o capa cliente, implementada como un applet o aplicación JAVA que puede ejecutarse desde cualquier máquina con acceso a Internet que disponga de una máquina virtual java. Este cliente se puede conectar a agentes JAVA distribuídos que funcionan como servidores, cada uno encargado de monitorear a un conjunto de agentes SNMP pertenecientes a una red local y que en general están ejecutando en dispositivos diferentes a los del agente JAVA.

### A.1 El agente

Como ya se ha descripto un agente JMX contiene un servidor de MBeans -MBean *Server*- que actúa como intermediario entre un conjunto de MBeans y la aplicación de Gerenciamiento. El agente contiene a los MBeans que modelan a los recursos gerenciados y servicios que provee el agente y disponen también de al menos un Adaptador de Protocolo (*protocol adapter* o PA) o Conector (*connector* o C).

### A.1.1 Clases MBeans

Como ya se ha descripto en el capítulo 5, los *MBeans* son objetos java que implementan una interface específica y cumplen con un determinado patrón de diseño. Estos requerimientos que deben cumplir los *MBeans* formalizan la representación de la ***interface de gerenciamiento*** del recurso en el *MBean*. La interface es la información que necesita una aplicación manager para poder operar con el recurso.
A continuación se muestra la interface **WrapperSNMPMBean** que define las operaciones SNMP que pueden ejecutarse remotamente sobre el agente JAVA. Por convención, una interface MBean interface tiene el mismo nombre de la clase que la implementa más el sufijo MBean. La clase que implementa esta interface se llama **WrapperSNMP y** tiene el comportamiento necesario para consultar a los agentes SNMP.

```
package cafetin.agente.mbeans;

import javax.management.snmp.SnmpOid;
import javax.management.snmp.SnmpVarBindList;
import javax.management.snmp.manager.SnmpPeer;
import javax.management.snmp.manager.SnmpRequest;

public interface WrapperSNMPMBean {
 public SnmpVarBindList snmpGetRequest(SnmpVarBindList l, SnmpPeer peer);
 public SnmpVarBindList snmpGetNextRequest(SnmpVarBindList l, SnmpPeer peer);
 public SnmpVarBindList snmpGetBulkRequest(SnmpVarBindList l, SnmpPeer peer);
 public SnmpVarBindList snmpSetRequest(SnmpVarBindList l, SnmpPeer peer);
 public SnmpVarBindList snmpGetPollRequest(SnmpVarBindList l, SnmpPeer peer);
 public SnmpRequest snmpInformRequest(SnmpVarBindList l, SnmpPeer peer,
                                      SnmpOid oid);
}
```

**Código 8.1 Interface WrapperSNMPBean (Servicios del SNMP)**

El **Código 8.2** muestra la clase `WrapperSNMP` la cual implementa la interface `WrapperSNMPBean` adoptando a la convención de nombres de JMX. Esta clase funciona como un wrapper para las operaciones de SNMP, brinda las funcionalidades de SNMP pero a través de JAVA pudiendo extender su funcionalidad. Esta clase tiene un constructor que crea una instancia de la `OidTable`, utilizada para trasladar nombres simbólicos en OIDs para facilitar las operaciones de SNMP.

```
public class WrapperSNMP implements WrapperSNMPMBean {
    private SnmpOidDatabaseSupport oidDB = new SnmpOidDatabaseSupport();

    public WrapperSNMP() {
       SnmpOid.setSnmpOidTable(oidDB);
       SnmpOidTableSupport oidTable = new RFC1213_MIBOidTable();
       oidDB.add(oidTable);
    }
    public SnmpVarBindList snmpGetRequest(SnmpVarBindList list,
                                          ipAddress ipAgenteSNMP) {
       SnmpRequest request = null;
       SnmpVarBindList result = null;
       boolean completed = false;
       int errorStatus = 0;
       try {
            System.out.println("Se creara el Peer con el port: " + 161);
            SnmpPeer agent = new SnmpPeer(ipAgenteSNMP, 161);
            SnmpSession session = new SnmpSession("session_sys_agent");
            session.setDefaultPeer(agent);
            request = session.snmpGetRequest(null, list);
            completed = request.waitForCompletion(10000);
            if (completed == false) {
             System.out.println(" GET Request: "+request.toString()+" timed
             out" + request.getResponseVarBindList());
            } else {
                // Hay una respuesta. Se chequea si tiene error
                errorStatus = request.getErrorStatus();
                if (errorStatus != SnmpDefinitions.snmpRspNoError) {
                    System.out.println("Error del Get = " +
                    SnmpRequest.snmpErrorToString(errorStatus));
                     System.out.println("Error del get: "+errorStatus);
                } else
                     result = request.getResponseVarBindList();
           }
           session.destroySession();
        } catch (Exception ex) {ex.printStackTrace();
        }
        return result;
    }
    . . .
  }
```

**Código 8.2: Ejemplo de un MBean estándar –WrapperSNMPMBean-**

Otro ejemplo de MBean es el definido para controlar desde JAVA el cambio de los valores de los atributos de tipos `String` de la MIB –de la misma manera se modelan MBeans para los otros tipos de la MIB-. Este MBean implementa la interface `AttributoStringSNMPMBean` que se lista en el código 8.3.

```
package cafetin.agente.mbeans;
public interface AtributoStringSNMPMBean {
    public void setData(String data);
    public String getData();
}
```

**Código 8.3 Interface AtributoStringSNMPMBean (Valores de SNMP)**

El código 8.4 lista la clase `AtributoStringSNMP` que modela los atributos de tipo String de la MIB. Básicamente es utilizada para responde el valor de un atributo consultado alimentándose de los valores de la MIB. Asimismo setea un nuevo valor a un atributo, generando una notificación a todos aquellos *listeners* que registraron interés en este evento de ***cambio de valor de un atributo*** –ver también código 8.5 y 8.6- .

```
package cafetin.agente.mbeans;
import javax.management.*;

public class AtributoStringSNMP extends NotificationBroadcasterSupport
                                implements AtributoStringSNMPMBean {
 private SnmpPeer peerSNMP = null;
 private String data = null;
 private SnmpVarBindList datax = new SnmpVarBindList();

 public AtributoStringSNMP(SnmpPeer peerSNMP, String data) {
        try {
            datax.addVarBind(data);
            this.peerSNMP = peerSNMP;
        } catch (SnmpStatusException e) {
            System.out.println("Error de Nombre de atributo ");
        }
}
public void setData(String data) {
  AttributeChangeNotification notification =
    new AttributeChangeNotification(this, 1,
     System.currentTimeMillis(),
    "el valor de \"data\" del objeto AtributioStringSNMP ha cambiado",
    "data", "String", this.getData(), data);
     this.data = data;
     sendNotification(notification);
}

public String getData() {
  SnmpVarBindList result = null;
  SnmpRequest request = null;
  boolean completed = false;
  int errorStatus = 0; SnmpPeer agent=null;
 try {
   agent = new SnmpPeer(peerSNMP.getDestAddr().getHostAddress(), 161);
   SnmpParameters params = new SnmpParameters("public", "public");
   agent.setParams(params);
   SnmpSession session = new SnmpSession("session_sys_agent");
   session.setDefaultPeer(agent);
   request = session.snmpGetRequest(null, datax);
   completed = request.waitForCompletion(10000);
   if (completed == false) {
       System.out.println("GET Request:"+request.toString()+
                    "timed out" +request.getResponseVarBindList());
    } else {
     errorStatus = request.getErrorStatus();
     if (errorStatus != SnmpDefinitions.snmpRspNoError)
            System.out.println("Error del Get");
        else
             result = request.getResponseVarBindList();
     }
    session.destroySession();
    data = result.varBindListToString();
 } catch (Exception ex) {
        ex.printStackTrace();
 }
 return data.substring(data.indexOf("Value : ")+8,data.length());
 }
}
```

**Código 8.4 Ejemplo de un MBean estándar – AtributoStringSNMP**

### A.1.2 Clases Listeners

Como se ha descripto en el capítulo 5, la especificación JMX define un modelo genérico de notificación basado en el Modelo de Eventos Java[1]. Este modelo permite que los MBeans, llamados **broadcasters**, propaguen notificaciones. Las notificaciones pueden ser emitidas por las instancias MBean o por el Server MBean. Otros MBeans u objetos, interesados en eventos que produce un MBean deben registrarse sobre él para ser notificados cuando un determinado evento ocurra; estos objetos son llamados **listeners**.

El modelo permite que un *listener*, se registre una única vez y reciba todas las notificaciones que ese MBean pueda emitir (hasta tanto no se quite esa registración). Además, un objeto *listener* puede registrarse en múltiples MBeans así como en un MBean se pueden registrar múltiples *listeners*.

El **Código 8.5** muestra una clase *listener* que implementa la interface **`NotificationListener`** para recibir **notificaciones emitidas por un MBean** cuando el valor de uno de sus atributos cambia. Esta clase hace uso del tipo de notificación predefinido de JMX, llamado **`AttributeChangeNotification`**, las cuales contienen el valor Viejo y el valor nuevo del atributo que cambió.

```
package cafetin.agente;

import javax.management.Notification;
import javax.management.NotificationFilter;
import javax.management.NotificationListener;
import javax.management.AttributeChangeNotification;

public class ChangeValueListener implements NotificationListener {

public  void handleNotification(Notification notification, Object obj) {
  AttributeChangeNotification attributeChange = null;

  if (notification instanceof AttributeChangeNotification) {
   attributeChange = (AttributeChangeNotification) notification;
   System.out.println("Notificación de cambio del Valor de un Atributo");
   System.out.println("Atr. Observado:"+attributeChange.getAttributeName());
   System.out.println("Viejo Valor: " + attributeChange.getOldValue());
   System.out.println("Nuevo Valor: " + attributeChange.getNewValue());
  }
 }
}
```

**Código 8.5: Definición de un *Listener* que informa cambio en el valor de un atributo**

Después de implementar una clase que escucha por notificaciones –en este caso de tipo **`AttributeChangeNotification-`**, se la debe registrar una instancia de la misma en el objeto que está interesado. El código 8.6 registra un *listener* directamente en un MBean llamado **`AtributosStringSNMPMBeans`** (descripta en el Código 8.4).

**Nota**: En este ejemplo del código 8.6 la registración del *listener* se hace en forma local, el propio agente crea el MBean, el Listener y lo registra. Sin embargo, la interface **`MBeanServerConnection`** –analizada en el código 8.10- también provee el método **`addNotificationListener`** que le permite a una aplicación cliente registrar un *listener* en un MBean situado en un agente remoto. En este caso, el conector

[1] Modelo de delegación de Eventos: la administración del evento es delegado desde la fuente del evento a uno o más objetos *listeners*.

servidor se encargará de propagar las notificaciones al conector cliente y de ahí a la clase *listener*.

```
private void inicializarAtributoString_Listener(){

 try {
   atributoStrName = new ObjectName(domain + ":name=AttributoStringSNMP");
   attributoStr = new AtributoStringSNMP();
   mbeanServer.registerMBean(attributoStr, atributoStrName);

   // Registro un Listener directamente en el MBean
   listenerChange = new ChangeValueListener();
   Filter filter = new AttributeChangeNotificationFilter();
   filter.enableAttribute("data");
   attributoStr.addNotificationListener(listenerChange,filter,atributoStrName);
 } catch (MalformedObjectNameException e) {
        e.printStackTrace();
 } catch (Exception e) {
        e.printStackTrace();
 }
}
```

**Código 8.6: Registración de un *Listener* en un MBean**

El código que registra el *listener* hace uso de un filtro disponible en la API para informar solamente si se produce una notificación de tipo **`AttributeChangeNotification`** y si el atributo que cambia es **`data`**. Esto hace al código más eficiente, porque si no se usa filtro, el *listener* informa sobre los cambios de todos los atributos del MBean.

En los párrafos anteriores se ha analizado la emisión de notificaciones directamente por parte de MBean que modelan los recursos gerenciados. A continuación se describe elmecanismo que debe utilizarse cuando se quiere que las **notificaciones sean emitidas por un Monitor** particular.

Las clases *Listener*s son suficientes cuando se necesitan detectar situaciones simples como el cambio del valor de un atributo; las mismas pueden ser instanciadas y registradas en el MBean en el cual se está interesado. Sin embargo, para situaciones donde se necesita un mecanimso más complejo de notificación o donde es necesario monitorear y correlacionar cambios en los valores de varios atributos, la API JMX provee Monitores. Los monitores –espacificados en el paquete **`javax.management.monitor`** de J2SE 5.0,-, como ya se ha explicado antes, también son MBeans que permiten comparar varios datos y enviar notificaciones bajo circunstancias específicas. Cada monitor consulta a los MBeans de acuerdo a la periodicidad[2] que se define el monitor para detectar situaciones espaciales.
El código 8.7 muestra un método que crea una instancia de **`CounterMonitor`** y lo configura con los valores recibidos como parámetro en el objeto **`Monitoreo`** y obtenidos del formulario visual -Figura 6.8- disponible en la aplicación cliente. Este objeto **`Monitoreo`** tiene información sobre el atributo a monitorear y los valores que se configurarán en el monitor como *threadhold*, *offset*, período de granularidad, etc.
El código muestra también la registración remota –desde el cliente en el agente- del Monitor y de un listener asociado.

[2] Periodicidad es el intervalo de tiempo entre las observaciones del valor de un atributo de un *MBean*

```
private void inicializaCounterMonitor(Monitoreo m) {

  ObjectName attributesSnmpName = null;
  ObjectName counterMonitorName = null;
  Object[] params = new Object[0];
  String[] signature = new String[0];

  // Creo un CounterMonitor MBean y lo agrego al MBeanServer
  try {
     System.out.println("\n>>> CREO un CounterMonitor REMOTAMENTE");
     counterMonitorName =
                  new ObjectName(domain+":name=CounterMonitor");
     mBeanCon.createMBean("javax.management.monitor.CounterMonitor",
                  counterMonitorName);  //deben ser nombres únicos

      // Creo el objeto a Monitorear y lo registro en el MBeanServer
     attributesSnmpName = new ObjectName(domain + ":name=AttributesSNMP");
     AttributesSNMP attributesSnmp = new AttributesSNMP();
     mBeanCon.createMBean("cafetin.agente.mbeans.AttributesSNMP",
                  attributesSnmpName);

     // Configuro los atributos del monitor
     monitorAttributes = new AttributeList();
     Attribute observedObjectAttribute = new Attribute("ObservedObject",
                                                 attributesSnmpName);
     monitorAttributes.add(observedObjectAttribute);
     Attribute observedAttribute = new Attribute("ObservedAttribute",
                                                       "Cambios");
     monitorAttributes.add(observedAttribute);
     Integer threshold = m.getThreshold();
     Attribute thresholdAttribute =
     new Attribute("Threshold", threshold);
     monitorAttributes.add(thresholdAttribute);
     Integer offset = = m.getOffset();
     Attribute offsetAttribute = new Attribute("Offset", offset);
     monitorAttributes.add(offsetAttribute);
     Attribute notifyAttribute = new Attribute("Notify", new Boolean(true));
     monitorAttributes.add(notifyAttribute);
     Attribute granAttribute =new Attribute("GranularityPeriod",m.getPerio());
     monitorAttributes.add(granAttribute);

     // Seteo los atributos que tiene que monitorear el Monitor
     monitorAttributes = mBeanCon.setAttributes(counterMonitorName,
                                              monitorAttributes);
     //Instancio y registro mi listener con el monitor
     ThreadholdListener listener = new ThreadholdListener();
     mBeanCon.addNotificationListener(counterMonitorName,listener,null,null);
     mBeanCon.invoke(counterMonitorName, "start", params, signature);

     // Espero por notificaciones
     synchronized (listener) {
     try {
          listener.wait();
     } catch (InterruptedException ignore) {
           System.out.print("Se interrumpe");
     }
  } catch (Exception e) { // por simplicidad dejo solo esta excepción
    e.printStackTrace();
  }
}
```

**Código 8.7: Registración remota de un Monitor y de un Listener en un agente**

Los listener que se registran en un Monitor son iguales a los que se registran en cualquier otro MBean, son clases que deben implementar la interface **NotificationListener** y en consecuencia darle comportamiento al método **handleNotification(Notification notification,Object handback**). El **Código 8.8** muetra un ejemplo de listener que emite una notificación cuando el valor observado alcanza un valor umbral o *threadhold*.

```
public class ThreadholdListener implements NotificationListener {

public void handleNotification(Notification notification,Object handback){

    MonitorNotification notif = (MonitorNotification) notification;
    CounterMonitor monitor = (CounterMonitor)notif.getSource();
    String type = notif.getType();
    try {
      if (type.equals(MonitorNotification.OBSERVED_OBJECT_ERROR)) {
                System.out.println("\n\t>> " +
                                   notif.getObservedObject() +
                                   " no está registrado en el Agente");
                System.out.println("\t>> Parando al CounterMonitor...\n");
                monitor.stop();
      } else if
       (type.equals(MonitorNotification.OBSERVED_ATTRIBUTE_ERROR)) {
                System.out.println("\n\t>> " +notif.getObservedAttribute()
                                  +" atributo no está contenido en el " +
                                  notif.getObservedObject());
                System.out.println("\t>> Parando al CounterMonitor...\n");
                monitor.stop();
            } else if
        (type.equals(MonitorNotification.OBSERVED_ATTRIBUTE_TYPE_ERROR)) {
                System.out.println("\n\t>> The type of " +
                                   notif.getObservedAttribute() +
                                   " attribute is incorrecto");
                System.out.println("\t>> Parando al CounterMonitor...\n");
                monitor.stop();
            } else if (type.equals(MonitorNotification.THRESHOLD_ERROR)) {
                System.out.println("\n\t>> El threshold no es de tipo
                                                        <Integer>");
                System.out.println("\t>> Parando al CounterMonitor...\n");
                monitor.stop();
            } else if
             (type.equals(MonitorNotification.THRESHOLD_VALUE_EXCEEDED)) {
                         System.out.println("\n\t>> " +
                         notif.getObservedAttribute() +
                         "Ha alcanzado el threshold\n");
             NotificarCliente.notificar("Threadhold excedido",
             attributeChange.getOldValue(),attributeChange.getNewValue());

     } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
     }
  }
}
```

**Código 8.8: Una clase Listener que informa que se ha alcanzado un *threadhold***

### A.1.3 El método main() del agente JMX

El agente cafeto es un aplicación de escritorio que no tiene interfaz de usuario gráfica. Tiene un método **main** que instancia un objeto **AgenteCafeto**, crea una conexión al agente SNMP –debería hacer esto para cada uno de los agentes que gerencia- y

registra dos adaptadores de protocolo y un conector JMX que permitirá a la aplicación de gereciamiento establecer una conexión remota al agente JAVA.

Para ejecutar el agente se lo puede hacer de tres maneras según la arquitectura que debe gerenciar:
- **java AgenteCafeto** (significa que el agente SNMP está en el mismo lugar que el agente JAVA, se establecerá una conexión desde el agente java al agente SNMP usando localhost).
- **java AgenteCafeto 192.168.1.100** (la IP de un único agente SNMP que debe gerenciar)
- **java AgenteCafeto agentes.txt** (en el archivo están las IP de los agentes SNMP que monitoreará este agente JAVA).

```
package cafetin.agente;

public class AgenteCafeto{
   . . .
   public static AgenteCafeto2011 getDefault() throws Exception {
       if (agente == null) {
           agente = new AgenteCafeto();
       }
       return agente;
   }

   public static void main(String[] args) throws Exception {
        // se analiza args
        agente = AgenteCafeto.getDefault();
        trace("Se terminó de inicializar el agente, ya está Listo!!!!");
   }

   private AgenteCafeto() {
     // Esto se debería hacer para cada uno de los agentes SNMP que gerencia
      mbeanServer = MBeanServerFactory.createMBeanServer();
      domain = mbeanServer.getDefaultDomain();
      try {
          peerSNMP = new SnmpPeer(host, portSnmp);
      } catch (UnknownHostException e) {
           System.out.println("No se puede conectar al agente SNMP: " + e);
      }
      this.inicializarConectorJMX();  // Se crea el Conector Server JMX
      this.inicializarHTMLAdaptor();  // Se crea Adaptador HTML
      this.inicializarSNMPAdaptor();  // Se crea Adaptador SNMP
   }
    . . .
}
```

**Código 8.8: La clase AgenteCafeto**

Un método importante para que las aplicaciones de gerenciamiento puedan conectarse con el agente es el método listado en el **Código 8.9** **inicializarConectorJMX()**. Este código crea un conector RMI servidor a través a partir del cual, las aplicaciones clientes podrán ejecutar operaciones en forma segura y remota sobre los MBeans del agente JAVA. La instanciación de un objeto **JMXServiceURL** permite tener una referencia a la url del agente JAVA.

```
package cafetin.agente;

public class AgenteCafeto {
   . . .
   private void inicializarConectorJMX() {
     try {
      System.out.println("Se crea un RMI registry en el port " + portRMI);
      try {
          LocateRegistry.createRegistry(portRMI);
      } catch (RemoteException e) {
          System.out.println(e);
      }
      String hostname = InetAddress.getLocalHost().getHostName();
      String url =
       "service:jmx:rmi:///jndi/rmi://"+hostname+ ":"+ portRMI +"/cafetin";

      JMXServiceURL jmxServiceURL = new JMXServiceURL(url);

     // Se genera agrega a un objeto Map los datos para autenticación/acceso
      SslRMIClientSocketFactory csf = new SslRMIClientSocketFactory();
      SslRMIServerSocketFactory ssf = new SslRMIServerSocketFactory();
      env.put(RMIConnectorServer.RMI_CLIENT_SOCKET_FACTORY_ATTRIBUTE,csf);
      env.put(RMIConnectorServer.RMI_SERVER_SOCKET_FACTORY_ATTRIBUTE,ssf);
      env.put("jmx.remote.x.password.file", "password.properties");
      env.put("jmx.remote.x.access.file", "access.properties");
      JMXConnectorServer cs =
      JMXConnectorServerFactory.newJMXConnectorServer(jmxServiceURL,
                                                      env, mbeanServer);
     // Se arranca el conector SERVER
       cs.start();
     } catch (Exception e) {
       e.printStackTrace();
     }
  }
  . . .
}
```

**Código 8.9: La clase AgenteCafeto -continuación-**

El código también muestra la definición de dos socket seguros basados en Secure Sockets Layer o SSL, uno cliente (csf) y uno servidor (ssf) que son guardados en un objeto **`Map`** para controlar los accesos al **`MbeanServer`**. La clase **`SslRMIClientSocketFactory`** es usada por RMI para obtener socktes vía SSL. Se definen dos archivos de propiedades con las contraseñas y con los roles y los niveles de acceso. Estos archivos los usa el conector server para autenticar al cliente (**`password.properties`**) y para autorizarlo (**`access.properties`**).
Luego se crea el conector server con el método de clase **`newJMXConnectorServer`** de la clase **`JMXConnectorServerFactory`** y se lo arranca con el método **`start()`**. De manera el MBeanServer solamente ejecutará las operaciones sobre los MBeans después de hacer una autenticación exitosa.

Archivo **`access.properties`**

Este archivo permite controlar los accesos remotos a los recursos del agente a través de la API JMX. Este archivo define los accesos permitidos para los diferentes roles. Una entrada consiste en el nombre de un rol y el nivel acceso asociado. El nombre del rol es cualquier *string* y el nivel de acceso es uno de los siguientes:

- ***readonly*:** permite el acceso para leer los atributos de MBeans. Para el monitoreo, esto significa que un cliente remoto con este nivel de acceso puede leer pero no puede cambiar atributos.

- ***readwrite*** permite el acceso para leer y escribir los atributos de MBeans. Este acceso se le pondría a usuarios confiables.

```
monitor readonly
administrador readwrite
```

Cada role debe tener al menos una entrada con el acceso. Si un rol no tiene entrada entonces no tiene acceso. Si un rol tiene múltiples entradas para un mismo rol entonces se usa la última entrada.

Archivo **password.properties**

Este es el archivo de autenticación para la API Remota JMX. Este archivo define las contraseñas para los diferentes roles. Una entrada consiste de un nombre de rol y una contraseña asociada. El nombre del rol y la contraseña son *strings* sin espacios en blanco. Notar que las contraseñas aparecen como texto simple por lo que no se recomienda usar contraseñas valiosas.

```
monitor mon!t
administrador adm!n
```

Por seguridad es conveniente restringir el acceso o especificar otro menos accesible.

## A.2 La aplicación de gerenciamiento

La aplicación de gerenciamiento o gerente puede ejecutar como *applet* o aplicación de escritorio y dispone de funcionalidades limitadas desarrolladas con propósito de pruebas.

### A.2.1 Comunicación con el Agente JAVA

Como ya se ha analizado en los capítulos anteriores una aplicación de gerenciamiento puede ejecutar operaciones sobre los MBeans del agente a través de los conectores apropiados una vez que se establece la conexión.
El **Código 8.10** muestra la lógica de conexión desde un cliente o aplicación de gerenciamiento al agente.

*Conectando JMX Cliente y Servidor*
Como puede observarse en el Código 8.10 la aplicación manager define una variable **u** de tipo **JMXServiceURL**, que representa la ubicación en la cual el conector cliente (ubicado en el manager) espera encontrar al conector server (ubicado en el agente). Esta url le permite al conector cliente recuperar el conector server RMI y establecer una conexión con el conector cliente, **jmxc**, como una instancia de **JMXConnector**, creada usando el **método connect()** del **JMXConnectorFactory**.

*Conectando la aplicación de gerenciamiento al MBean Server remoto*
Una vez establecida la conexión entre los Conectores RMI, el cliente debe establecer una conexión con el *MBeanServer* para poder interactuar con los *MBeans* registrados en él. Para esto se utiliza el método **getMBeanServerConnection**. El conector cliente ahora está conectado al **MBeanServer** y puede registrar *MBeans* y ejecutar operaciones sobre los *MBeans* existentes.

*Ejecutando operaciones sobre MBeans Remotos vía Proxies*
El cliente accede al **WrapperSNMP** en el *MBeanServer* a través de la conexión **mBeanCon** creando un proxy, local al cliente y emulando al *MBean* (**WrapperMBean**) remoto. Para hacerlo se utiliza un método de clase de JMX.

Una vez que el cliente JMX ha obtenido toda la información necesaria y realizó todas las operaciones sobre los *MBeans* del agente JMX remoto, la conexión debe ser cerrada utilizando el método **close()**.

```
/*
* Si se conecta un administrador los valores en los parámetros serían:
* usuario="administrador" y password="adm!n"
*/
public void conectar(String hostname, String portRMI, String usuario,
                                                      String password) {

  String url=null;
  JMXServiceURL u=null;
  try {

    url="service:jmx:rmi:///jndi/rmi://"+hostName+":"+portRMI+"/cafetin";

    // Se establece una conexión segura
    u = new JMXServiceURL(url);
    HashMap env = new HashMap();
    String[] credentials = {usuario, password};
    env.put("jmx.remote.credentials", credentials);
    JMXConnector jmxc1 = JMXConnectorFactory.connect(u, env);

    MBeanServerConnection mBeanCon = jmxc.getMBeanServerConnection();
    String domain = mBeanCon.getDefaultDomain();
    . . .
    WrapperSNMP wrapper = null;
    ObjectName wrapperObjName = null;
    SnmpVarBindList result = null;
    try {
      wrapperObjName = new ObjectName(domain + ":class=WrapperSNMP");
      WrapperSNMPMBean mbeanProxy = JMX.newMBeanProxy(
                          mBeanCon,
                          wrapperObjName,
                          WrapperSNMPMBean.class, true);
      result = mbeanProxy.snmpGetRequest(getList);

      if (result != null) {
        Iterator li = result.iterator();
        while (li.hasNext()) {
              System.out.println("Lista: " + li.next());
        }
      } else
        System.out.println("Error no se puede invocar el snmpGetRequest");

    } catch (MalformedObjectNameException e) {
        System.out.println("Nombre mal formado " + e);
    }
}
```

**Código 8.10: Conexión desde un cliente JMX a un agente JMX**

## A.3 El compilador mibgen

El JDMK provee un compilador de MIB SNMP **mibgen**. Este es un compilador de MIB SNMP basado en JAVA que toma como entrada una especificación de una MIB, la parsea y genera las siguientes clases:
- Una clase que mapea nombres simbólicos con identificadores de objetos de las variables de MIB. Esta clase es útil tanto para los agentes como para los managers.
- Una clase que representa a la MIB completa.

- Clases que representan los grupos de las MIBs
- Clases que representan las tablas SNMP.
- Clases que representan los tipos enumerados SNMP.

Todas las clases generadas por el **mibgen** deben ser provistas de una implementación definitiva.
Entre las clases generadas se encuentra la clase xxx que extiende la clase SnmpMib, la cual es una abstracción lógica de una MIB SNMP. El adaptador SNMP usa esta clase para implementar el comportamiento del agente. En el caso de esta tesis se proveyó como entrada xxx.txt y

El compilador también genera un archivo JAVA que contiene el código requerido para representar las variables de la MIB.

# REFERENCIAS


**[Ahmed, M.]** M. Ahmed, K. Tesink, *Definitions of Managed Objects for ATM Management Version 8.0 using SMIv2,* IETF 1695, Agosto 1994.

**[Baecker, R.]** Ronald M. Baecker, *Reading in Groupware and Computer-Supported Cooperative Work, Assistant HUMAN-HUMAN COLLABORATION*, Morgan Kaufman Publishers, ISBN=1-55860-241-0.

**[Bauer C.]** Christian Bauer, Gavin King, *Hibernate in Action*, Manning, ISBN=9781932394153,

**[Case, J.; RFC1157]** J. Case, M. Fedor, M. Schoffstall, J. Davin, *A Simple Network Management Protocol (SNMP)*. IETF RFC 1157, Mayo 1990.

**[Case, J.; RFC1442]** J. Case, K. McCloghrie, M. Rose, *Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)*. IETF RFC 1442, abril 1993.

**[Case, J.; RFC1905]** J. Case, K. McCloghrie, M. Rose, Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2). IETF RFC 1905, enero 1996.

**[Case, J.; RFC1906]** J. Case, K. McCloghrie, M. Rose, Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2). IETF RFC 1906, enero 1996. `

**[Case, J.; RFC2570]** J. Case, R. Mundy, D. Partain, Introduction to Version 3 of the Internet-standard Network Management Framework). IETF RFC 2570, abril 1999. `

**[Clemm, A.]** Alexander Clemm, Network Management Fundamentals, CISCO Press 2007. ISBN: 1-58720-137-2.

**[CMU-SNMP]** CMU-SNMP, Network Group Software, librerías desarrolladas en Carnegie Mellon, accesible en http://www.net.cmu.edu/groups/netdev/software.html

**[Coltun, R.]** R. Coltun, OSPF Version 2 Management Information Base. IETF RFC 1253, Agosto 1991.

**[Geary, D; a]** David M. Geary, *Graphics JAVA, Mastering the JFC*, Volume I, AWT, 3rd edition. SUN Microsystem, ISBN=0-13-079666-2.

**[Geary, D; b]** David M. Geary, *Graphics JAVA, Mastering the JFC*, Volume II, Swing, 3rd edition. SUN Microsystem, ISBN=0-13-079667-0

**[Hortsmann, C.]** C. Hortsmann, G. Cornell, Core Java 2: Volume I–Fundamentals, Prentice Hall PTR, ISBN: 0-13-089468-0. Fifth Edition, Diciembre 2000.

**[ISO 8825]** *ASN.1 encoding rules*: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), ISO/IEC 8825, diciembre 2008 (última revisión).

**[ISO 8824]** *Abstract Syntax Notation One (ASN.1)*: Parameterization of ASN.1 specifications, ISO/IEC 8824 (estándares que describen las reglas de codificación de ASN.1), diciembre 2002.

**[JavaBeans]** Java Beans Specifications for Java 2, accesible en http://www.oracle.com/technetwork/java/javase/index-138195.html

**[JDMK]** Java Dynamic Management Kit, http://java.sun.com/products/jdmk/index.jsp

**[JMX]** Java Management Extensions Specification, version 1.4, http://download.oracle.com/javase/7/docs/technotes/guides/jmx

**[JMXRes]** *Instrumenting Your Resources for JMX Technology*, accesible en http://docs.oracle.com/javase/7/docs/technotes/guides/jmx/overview/JMXoverviewTOC.html, abril 2012.

**[JSR3]**: Java Specification Requests 3, Java Management Extensions (JMX) specification, accesible en http://jcp.org/en/jsr/detail?id=003

**[JSR160]**: Java Specification Requests 160, Java Management Extensions (JMX) Remote API specification, accesible en http://jcp.org/en/jsr/detail?id=160

**[JSR255]**: Java Specification Requests 255, Java Management Extensions (JMX) Specification, version 2.0, accesible en http://jcp.org/en/jsr/detail?id=255

**[Leinwand A, Conroy, K]** Allan Leinwand y Karen fang Conroy de CISCO System. *Network Management, A Practical Perspective*. Addison Wesley, ISBN=0-201-60999-1, 1998.

**[McCloghrie, K.]** K. McCloghrie, M. Rose, *A Management Information Base for Network Management of TCP/IP-based internets: MIB-II Simple Network Management Protocol (SNMP)*. IETF RFC 1213, Mayo 1991.

**[Postel J.]** J. Postel, *Internet Control Message Protocol,* IETF RFC 792, Septiembre 1991.

**[Rose, M.; RFC1155]** M. Rose, K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based Internets*. IETF RFC 1155, Mayo 1990.

**[Rose, M.; RFC1156]** M. Rose, K. McCloghrie, *Management Information Base for Network Management of TCP/IP-based internets*. IETF RFC 1156, Mayo 1990.

**[Stalling W.]** Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards, William Stallings, The Internet Protocol Journal, Volumen I, Número 3. Accesible en http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-3/ipj_1-3.pdf.

**[Stefik, M]** M Stofik, D. Bobrow, G. Foster, S. Lanning, *WYSIWIS Revised: Early experiences with Multiuser Interfaces*. Xerox Palo Alto Research Center. Accesible en: http://itee.uq.edu.au/~comp3505/Resources/Course%20overview/_papers/p147-stefik.pdf

**[Waldbusser, S.]** S. Waldbusser, *Remote Network Monitoring Management Information Base*. IETF RFC 1757, Febrero 1995.

**[Willis, S.]** S. Willis, J. Burruss, *Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2*. IETF RFC 1657, Julio 1994.

**[WebNMS]** Web Network Management System, AdventNet. Sitio oficial accesible desde http://www.webnms.com/

**[Wijnen, B.]** B. Wijnen, R. Presuhn, View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). IETF RFC 3415, diciembre 2002.