Universidad de Zaragoza

PROYECTO FIN DE CARRERA
Ingeniería de Telecomunicación

Implementación automática de un agente SNMP a partir de la definición formal de su MIB

Juan Carlos Arribas Arribas

Director:
Nelia Lasierra Beamonte

Ponente:
Álvaro Alesanco Iglesias

Septiembre 2012
Curso 2011-2012

Agradecimientos

En primer lugar, quiero dar las gracias a ***Álvaro*** *por brindarme la idea y la posibilidad de realizar este Proyecto Fin de Carrera en un área que me llamó mucho la atención durante las clases en su asignatura. No quiero ni mucho menos olvidarme de* ***Nelia****. Ella me ha ayudado en todo momento, me ha sacado siempre que ha podido de los momentos en los que me quedaba atascado y también me ha dado ánimos y me ha metido presión para sacar siempre lo mejor de mí. Siempre te estaré muy agradecido.*

A mis padres y hermano, los cuales siempre me han apoyado y, sobre todo, me han aguantado en mis momentos de mayor estrés y han sido un pilar fundamental durante toda la carrera.

A mis amigos, por su interés y apoyo, por sus consejos, por regalarme el ocio y diversión aunque no os haya podido hacer mucho caso durante estas últimas semanas: ***Jose*** *y* ***Ricar****, y en especial,* ***Tere****, la cual ha tenido que soportar más que nadie mi estrés e historias aburridas de trabajo.*

A mis compañeros de la universidad, quienes siempre me han apoyado y me han dado tantos buenos momentos a lo largo de la carrera y también durante este proyecto que habéis hecho que no extrañara mi ciudad: ***Diego****,* ***Aguilar****,* ***Sergio****,* ***Fran****,* ***Lourdes****,* ***Sandra****,* ***Pablo****... y muchos otros que habéis estado conmigo en todo momento.*

“Implementación automática de un agente SNMP a partir de la definición formal de su MIB”

RESUMEN

En este proyecto se ha desarrollado un conjunto de herramientas que facilitan el desarrollo de agentes SNMP (Simple Network Management Protocol) así como de las MIBs (Management Information Base) que implementa a partir de su definición formal. El sitema consta de tres bloques independientes entre sí, pero que están diseñados para funcionar conjuntamente. El primer bloque consta de un interfaz gráfico con las herramientas necesarias para poder diseñar una MIB. El segundo bloque se encarga de transformar la definición formal de una MIB cualquiera en una estructura de tablas genéricas para su almacenamiento en una BBDD (base de datos). El último bloque permite desarrollar un agente SNMP para establecer comunicaciones SNMP con un gestor externo y así facilitar a este la gestión de la MIB implementada en la BBDD.

Para la utilización de todos estos bloques se han implementado tres interfaces (uno por bloque) para que su uso por el usuario sea fácil e intuitivo. El interfaz del primer bloque (*MIB Builder*) sirve para crear la definición formal de la MIB de forma gráfica para evitar el empleo por parte del usuario del lenguaje SMI (Structure of Managed Information). El interfaz del segundo bloque (*BBDD Creation*) permite al usuario introducir el archivo de texto generado anteriormente con la definición formal de la MIB, e introducir la información de configuración de la BBDD creada para que se genere la estructura de tablas genéricas en la BBDD que dan soporte a dicha MIB. Por último, el interfaz del tercer bloque (*Agente SNMP*) permite gestionar las cuentas de usuario que tendrán acceso al agente, además de establecer comunicaciones SNMP con cualquier gestor externo mediante las tres versiones que se definen en la arquitectura.

Finalmente, se ha realizado una fase de pruebas para verificar el correcto funcionamiento del sistema propuesto. Utilizando el primer bloque se han diseñado MIBs conocidas y nuevas demostrando así la funcionalidad del interfaz para obtener su definición formal. Para demostrar el correcto funcionamiento del segundo bloque se han implementado tanto MIBs nuevas como MIBs conocidas y posteriormente mediante la utilización de un gestor comercial se han establecido comunicaciones SNMP con el agente desarrollado y accedido a la información contenida en las MIBs creadas.

Se puede concluir que el conjunto de bloques desarrollado constituye un sistema integrado para el desarrollo de agentes partiendo de la idea inicial de una MIB, ya que se proporcionan todas las herramientas necesarias para conseguir este propósito. También constituye una manera de implementar agentes SNMP de una manera estándar donde la interoperabilidad con los gestores está garantizada.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Objetivos

1.1 Introducción

En la actualidad las tecnologías de la comunicación se han hecho imprescindibles en nuestra sociedad. Esto lo podemos ver por ejemplo, en la vida cotidiana de las personas, estando cada vez más necesitados de la redes sociales, también en las instituciones públicas, gestionando cada vez más servicios a través de Internet y más aún en el mundo empresarial, tanto en pequeñas como en grandes empresas. Cada día es más necesario estar presentes y ser conocidos en cualquier parte del mundo. Lejos quedan ya los días en los que el uso frecuente de Internet era minoritario entre la población. Por tanto se puede decir sin temor a equivocarnos, que estas redes son cada día más grandes. Al ser cada vez más grandes, hay más elementos presentes en dichas redes. Para que la red funcione correctamente, se necesita conocer el estado de la misma y monitorizar cierta información relevante de determinados elementos de la red (swichs, hubs, routers...). Para esto son necesarias pautas y reglas que permitan hacer este proceso de manera unificada y automatizada.

Por ello existen protocolos y arquitecturas de gestión que facilitan esta tarea a los gestores de la red. Estos protocolos permiten recopilar información de los equipos para conocer el estado de las redes y poder gestionar así de manera eficiente la misma. Este hecho permite ofrecer cada día servicios más amplios y mejorar a su vez la calidad del servicio ofrecido a los usuarios de la red. Hoy en día, la arquitectura SNMP [1] (Simple Network Management Protocol) es la más popular y estándar *de facto* implementada por la mayoría de fabricantes de equipos. La información que se puede gestionar de cada uno de los equipos controlados mediante SNMP se almacena en unas BBDD (bases de datos) que se llaman MIBs (Management Information Base). Dichas MIBs contienen la información que se desea gestionar en cada uno de los componentes que las llevan implementadas (routers, switchs, hubs...), por lo que el conocimiento de esta información nos permite conocer el estado del tráfico presente en la red.

Dadas las ventajas de gestión que ofrece SNMP, este proyecto fin de carrera se basa en el estudio y desarrollo de herramientas de esta arquitectura. Esta es hoy en día la más popular pero existen otras como pueden ser CORBA [2] (Common Object Request Broker Arquitecture) cuya característica principal es que esta arquitectura nos permite comunicar aplicaciones definidas en diferentes lenguajes y ejecutados en distintas plataformas de manera trasparente sin tener que realizar ningún proceso intermedio. Otro protocolo existente actualmente es NETCONF [3], el cual se basa en XML [4].

A pesar de la popularidad y simplicidad de la arquitectura SNMP para gestionar redes, la implementación de nuevas MIBs no es una tarea sencilla para los desarrolladores. Además, hoy en día no existe una organización interna estándar de las bases de datos que dan soporte a las MIBs y cada desarrollador diseña su propia base de datos de acuerdo a las características específicas de su MIB. SNMP puede utilizarse para gestionar otros dispositivos distintos a los típicos de una red de comunicaciones como un switch o un hub y permite por tanto que se pueda utilizar para otros servicios de gestión. Por lo tanto el estudio de la automatización de este proceso resulta de gran interés. Un ejemplo, es el uso en dispositivos médicos para el intercambio de información de un determinado equipo situado en casa de un paciente al centro hospitalario.

Una vez generada la estructura de tablas de la BBDD es necesario un módulo que facilite la interacción entre un agente SNMP y la BBDD creada. Este agente establecerá comunicaciones SNMP con un gestor externo. Una vez recibidos los mensajes de petición de este, el agente interactuara con su MIB (BBDD creada) para darle a dicho gestor la información solicitada.

La problemática de implementar MIBs en BBDD y agentes se traslada también a la definición formal de las mismas, ya que aunque existen diversos programas que te permiten generar el código de la definición formal de una MIB de una manera gráfica e intuitiva, dicho software no es de libre distribución y las licencias para su utilización son costosas para cualquier diseñador independiente que no trabaje en una gran empresa que se permita su adquisición. Por esto y porque es mucho más fácil crear la definición de la MIB con ayuda de su visualización gráfica, se propone implementar una herramienta que disponga de las funcionalidades de un MIB Builder, un BBDD Creation (algoritmo para el desarrollo de una BBDD genérica) y un agente que interactúe con este último modulo.

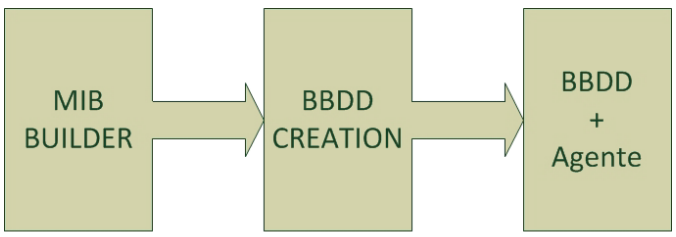

Figura 1.1: Proceso del problema a resolver.

En la Figura 1.1 se muestra el proceso a desarrollar. Lo que se pretende es ofrecer un entorno de gestión completo basado en SNMP: una vez diseñada la MIB con el MIB Builder, el BBDD Creation generará la estructura de la BBDD con la cual interaccionará el agente SNMP para ofrecer la información solicitada a un gestor externo SNMP.

1.2 Estado del Arte

Actualmente existen empresas que desarrollan MIB Builders como son MG-SOFT [5] y NuDESIGN [6]. Estos programas tienen una versión de evaluación del producto, con la cual se ha trabajado para investigar su funcionamiento con el fin de que nos sirviera de punto de partida a la hora de realizar nuestro propio MIB Builder. Una visualización de como son estos dos programas, se muestra en la Figura 1.2.

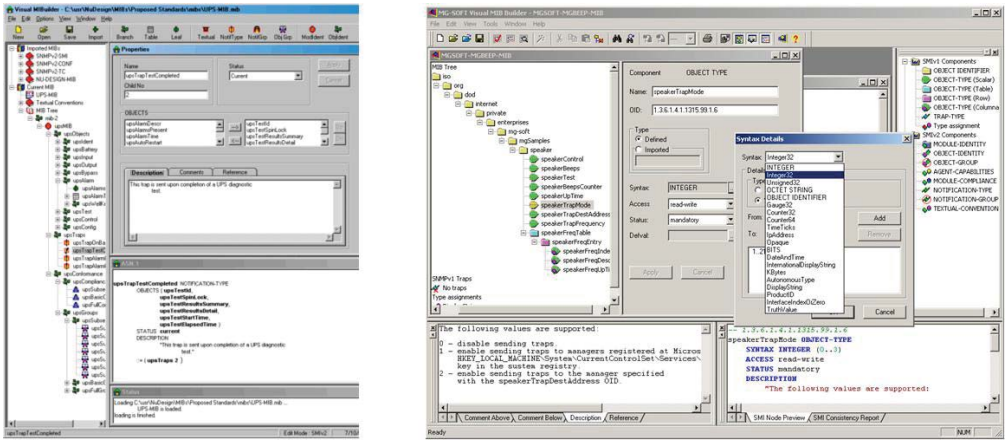

Figura 1.2: MIB Builder comerciales.

Ambos interfaces muestran una zona en la que se va dibujando el diagrama de hojas y ramas que componen la MIB, y otra zona donde se editan los parámetros de los componentes introducidos. También hay otro espacio dedicado a ir mostrando la sintaxis formal de la MIB actualizada con cada cambio realizado. Estas características que hemos destacado, han servido para inspirar el diseño del MIB Builder diseñado en este proyecto.

Previamente al diseño del algoritmo de automatización de MIBs se realizó una búsqueda de información sobre modelos de BBDD para la implementación de MIBs. La mayoría de las soluciones basadas en SNMP y el desarrollo de MIBs solo se centran en describir el diseño del contenido de la MIB pero ninguna se preocupa de como es la implementación de dicha MIB ni existe ninguna propuesta de como debe ser la interacción con el agente ([7]-[11]). Por ejemplo, en [7], se cuenta de la importancia de las MIBs como una parte del total de un NMS (Network Management System), debido a que las MIBs tienen la capacidad de proporcionar información relevante para conocer el estado de la red y por tanto facilitar su gestión. En [8], se presenta que el tipo de

BBDD (relacional u orientada a objetos) que se implementa en las MIBs es indiferente, pero tampoco detalla como es la implementación. En [9] y [10] se centran en detallar los campos de la MIB y la descripción del sisitema. Por tanto, se llegó a la conclusión de que en base a la bibliografía revisada no hay ninguna propuesta para la implementación de MIBs en BBDD y cada fabricante de dispositivos elabora la BBDD en función de cada dispositivo específico, sin seguir ningún tipo de reglas.

El desarrollo de agentes es importante ya que no solo se utilizan para la gestión de redes en entornos LAN ([9] y [11]) sino que se emplean para otras aplicaciones como pueden ser la de gestión de dispositivos médicos ([10]). Por ello su desarrollo es interesante. Actualmente existen herramientas que facilitan su desarrollo (SNMP Agent Builder [12]), pero no hay una herramienta que genere el agente junto con el módulo de interacción a su MIB.

1.3 Propuesta

En este proyecto se propone como objetivo principal el desarrollo de un sistema de gestión SNMP compuesto por tres módulos, cuyo núcleo central es el desarrollo de un algoritmo que implemente de manera automática la creación de la BBDD de una MIB genérica. Con todo esto, se pretende que su conjunto sirva para implementar el agente en un dispositivo partiendo de la base del diseño inicial de su MIB.

Los tres módulos que componen el sistema propuesto son los siguientes:

- MIB Builder (Módulo de diseño): Este primer módulo ayudará a crear la definición formal de una MIB de una manera gráfica e intuitiva de manera similar que los programas comerciales.

- BBDD Creation (Módulo de almacenamiento): Este segundo módulo es el más importante puesto que será el encargado de crear la BBDD partiendo de la definición formal de la MIB. Este programa debe permitir transformar la definición formal de cualquier MIB en una BBDD independientemente de su organización de hojas y tablas y que maximice la eficiencia en el acceso a la misma. Por ello se desarrollará un algoritmo para la lectura de MIBs y su relación con una estructura genérica en BBDD propuesta para su almacenamiento.

- Agente SNMP (Módulo de comunicaciones): Este tercer módulo constará de la implementación de un agente SNMP, y nos permitirá poder establecer las conexiones con un gestor externo a través de SNMP y así poder hacer pruebas con las que comprobar la robustez del algoritmo previamente creado. Para hacer más genérico el modo en que se puedan conectar al agente, el diseño de este agente dará soporte a las tres versiones (v1, v2c y v3) definidas en la arquitectura.

Para que todo este sistema global sea amigable al usuario final, se propone finalmente crear una serie de interfaces gráficos para interactuar con los programas creados.

Por tanto y a modo de resumen, tal y como se puede ver en la Figura 1.3 se propone generar un interfaz compuesto por tres herramientas diferentes, que se puedan usar también por separado, y que nos permita en primer lugar obtener la definición formal de una MIB, en segundo lugar, generar la BBDD de una MIB y finalmente poner en marcha el agente para que el sistema quede completo.

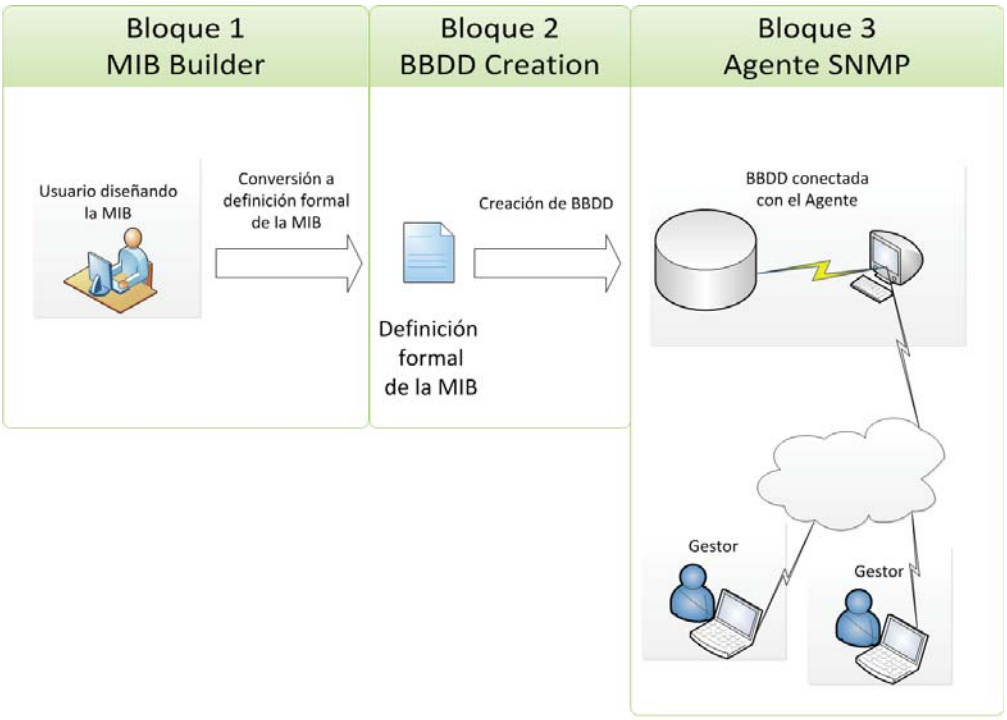

Figura 1.3: Esquema general propuesto y la interacción entre los bloques.

1.4 Objetivos

El objetivo principal de este proyecto es el desarrollo de una herramienta de gestión SNMP completa que incorpore un modulo para el desarrollo de MIBs, un módulo para la implementación automática de MIBs en BBDD y un agente SNMP que interactúe con dicha MIB y con el gestor externo. Para llevar a cabo la realización del proyecto se han completado los siguientes objetivos:

1. Para el módulo de diseño de MIBs:

   - Estudio y compresión del lenguaje SMI (lenguaje formal para la definición de MIBs) para conocer la base sintáctica de una MIB, conocer sus componentes y todas sus características.

- Desarrollo de un interfaz gráfico con el que el usuario interactúe para que pueda diseñar su MIB.
- Creación de un algoritmo que recoja la información introducida por el usuario y la transforme en el texto de la definición formal de la MIB diseñada.

2. Para el módulo de almacenamiento en BBDD:

   - Estudio y compresión del protocolo SMI para conocer la base sintáctica de una MIB, conocer sus componentes y características relevantes a tener en cuenta a la hora de almacenarlas en la BBDD.
   - Estudio y recopilación de los diferentes tipos de MIB existentes con el fin de proponer un sistema de almacenamiento genérico que sirva para todas ellas.
   - Proponer una organización de la información en diferentes tablas para tratar su gestión y almacenamiento en la BBDD.
   - Estudio del lenguaje SQL para poder crear y gestionar la BBDD.
   - Implementar y desarrollar un conjunto de clases Java para poder estructurar la información obtenida del fichero de texto introducido con la definición formal de la MIB.
   - Creación de un algoritmo que recopile toda la información almacenada en las clases Java creadas previamente y cree la BBDD a partir de ellas.
   - Desarrollo de un interfaz gráfico con el que el usuario pueda seleccionar la MIB a tratar y la información de control de su BBDD (nombre de usuario, password...).

3. Para el módulo de comunicaciones:

   - Estudio en profundidad de la arquitectura SNMP para adquirir los conocimientos necesarios para tratar el procesado de un mensaje o PDU (GET, GETNEXT, SET...), y poder dotar al sistema de la capacidad necesaria para su funcionamiento en las tres versiones existentes (v1, v2c y v3).
   - Estudio en profundidad de la arquitectura SNMP en su versión 3 para poder proporcionar al sistema de privacidad y seguridad.
   - Desarrollo de un algoritmo que procese las peticiones que nuestro agente reciba de un gestor e interactúe con la BBDD para proporcionarle el dato requerido.
   - Desarrollo de un programa que pueda interaccionar con la BBDD para introducir datos con permisos exclusivos para el agente.
   - Desarrollo de un interfaz gráfico con el que el usuario pueda interactuar con nuestro agente (administrar cuentas de usuario, puesta en marcha y detención del agente).

4. Finalmente se creará un único interfaz gráfico con el que interactúe el usuario, que contenga los tres interfaces creados en cada uno de los bloques.

1.5 Materiales

El lenguaje de programación utilizado para el desarrollo del trabajo ha sido JAVA, lenguaje orientado a objetos que ofrece un marco potente de programación basado en clases para el desarrollo de programas de gran eficiencia. Además, hemos realizado la interacción con la BBDD de forma sencilla a través de conexiones JDBC y la utilización de sentencias SQL.

Se han empleado los siguientes programas software, todos ellos de libre distribución (o en versiones de evaluación gratuitas):

- SDK Eclipse [13].
- MG SOFT MIB Browser [14].
- MySQL [15] (phpmyAdmin de Xamp).
- La API SNMP4j [16].

1.6 Organización de la memoria

La memoria está estructurada de la siguiente manera:

- **Capítulo 1: Introducción.** Es el capítulo actual y contiene una breve descripción del trabajo realizado, así como sus principales objetivos.
- **Capítulo 2: Características de la arquitectura SNMP y del lenguaje SMI.** Este capítulo contiene una descripción de las principales características del lenguaje formal que se emplea en las MIBs (el lenguaje SMI) y también se describen los elementos mas importantes de la arquitectura SNMP.
- **Capítulo 3: Desarrollo Tecnológico: Bloques implementados.** Este capítulo contiene una descripción de los bloques implementados: MIB Builder, BBDD Creation y Agente SNMP.
- **Capítulo 4: Pruebas y Resultados.** Este capítulo contiene un informe completo de los resultados obtenidos de las pruebas a las que se ha sometido al programa.
- **Capítulo 5: Conclusiones y líneas futuras.** Este es el último capítulo de la memoria y contiene las conclusiones que se han sacado en este proyecto y las posibles líneas futuras que se podrían seguir.

El contenido de los anexos incluidos es el siguiente:

- En el anexo A se tiene una lista de los acrónimos utilizados en esta memoria.
- En el anexo B se tiene la guía de usuario para utilizar el MIB Builder desarrollado correctamente.
- En el anexo C se tiene la descripción de las tablas implementadas para la creación de la BBDD de una MIB.
- En el anexo D se tiene la guía de usuario para introducir valores en la MIB por parte del agente implementado.
- El anexo E contiene los diagramas de flujo realizados para la implementación del procesado de paquetes que le lleguen al agente implementado.
- En el anexo F se tiene la guía de usuario para utilizar el interfaz Agente SNMP desarrollado correctamente.
- En el anexo G se incluye el diagrama de Gantt del proyecto que describe el reparto temporal entre las diferentes tareas que han conformado el mismo.

Capítulo 2

Características de la arquitectura SNMP y lenguaje SMI

El lenguaje formal que se usa para definir MIBs es el lenguaje SMI, el cual se ha tenido que estudiar en profundidad para la elaboración de este proyecto fin de carrera. En este capítulo se describen las características de dicho lenguaje y se presenta también la arquitectura SNMP. Se dan las claves teóricas para facilitar la comprensión del desarrollo de los modulos del entorno de gestión presentado en el proyecto.

2.1 La arquitectura SNMP

SNMP es la arquitectura de gestión de redes más utilizada en redes TCP/IP debido a su simplicidad y su escaso consumo de recursos. SNMP define un protocolo para el intercambio de información de gestión además de definir un formato para la representación de esa información y un marco para organizar sistemas distribuidos en gestores y agentes. La arquitectura SNMP puede ser descrita por los siguientes elementos claves:

1. Dos entidades SNMP: Una entidad gestionada (agente), y una entidad gestora (gestor).

2. Un conjunto de mensajes SNMP con los que funciona el protocolo de gestión de la red.

3. La MIB: BBDD del agente.

La primera versión de SNMP (SNMPv1) presentaba debilidades tanto en términos de seguridad de comunicaciones como en términos de eficiencia ya que no podía proporcionar en un mismo paquete grandes cantidades de información. La segunda versión (SNMPv2c) solucionaba el problema de entregar grandes cantidades de información en un mismo

paquete creando un nuevo tipo de mensaje (GetBulk), pero seguía teniendo los mismos problemas de seguridad. Así surge SNMPv3, como una evolución de SNMPv2c ya que en cuanto al comportamiento funcional es el mismo, lo que cambia es que añade seguridad y cifrado en la comunicación.

2.1.1 Entidades SNMP

La arquitectura SNMP se basa en la interacción de al menos dos entidades SNMP, un agente y un gestor (aunque pueden ser varios los gestores que intervengan). Cada una de estas entidades contiene un conjunto de módulos que interaccionan para proporcionar servicios. Cada entidad contiene un *SNMP engine* que es capaz de enviar y recibir mensajes SNMP. El papel de cada entidad SNMP será determinado por los módulos implementados en ella misma. Cada entidad debe actuar como agente, gestor o una combinación de ambas. SNMP permite a múltiples agentes interaccionar con múltiples gestores y esto nos permite tener múltiples gestores interaccionando con el mismo agente para intercambiar información.

- Agente: Es la entidad SNMP situada en el equipo que se va a monitorizar, sus misiones son recopilar y guardar información local, así como responder ante peticiones del gestor y enviar información de forma asíncrona cuando sucede algún evento. Toda esta información puede ser gestionada en el dispositivo y guardada en una base de datos externa diseñada llamada MIB.

- Gestor: Es la entidad SNMP encargada de pedir información y modificarla según las necesidades de funcionamiento de la máquina en la que reside el agente. El gestor o gestores de la red poseen un interfaz gráfico para poder emitir comandos y examinar en forma de tablas los datos que les llegan de los agentes. Estos sistemas incluyen como mínimo aplicaciones para monitorización, control de configuración y realización de informes.

El *SNMP engine* proporciona servicio entre la capa de transporte y las aplicaciones SNMP. Un ejemplo es que encapsula el PDU (Protocol Data Units) dentro del mensaje para transmitirlo y también de manera inversa, para procesar información.

2.1.2 Mensajes SNMP

El protocolo SNMP define una serie de mensajes para el intercambio de información, proporciona esencialmente cuatro tipos de mensajes:

1. **Get:** Permite al gestor recibir información del agente.

2. **Set:** Permite al gestor modificar algún valor en la MIB del agente.

3. **Trap:** Permite al agente enviar mensajes asíncronos al gestor.

4. **Inform:** Permite al gestor enviar algún mensaje de alerta a otro gestor.

Para implementar estos tipos de mensajes, SNMP especifica un conjunto de PDUs diferentes entre la comunicación agente-gestor. Estos PDUs están encapsulados en los mensajes SNMP con la cabecera adecuada para la versión del protocolo implementada.

- Mensajes enviados por el gestor al agente:
  - *GetRequestPdu:* petición de una o varias variables incluidas en el mensaje de respuesta Response.
  - *GetNextRequestPdu:* petición de la inmediatamente siguiente variable a la variable que mandamos y con valor no nulo, cuya respuesta se incluye también en un mensaje Response. Este comando se usa para recorrer la estructura en árbol de la MIB.
  - *GetBulkRequestPdu:* tipo de petición presente solo a partir de la versión 2 (SNMPv2c). Nos permite recibir una lista de variables consecutivas para cada variable que le mandamos nosotros. El número de variables consecutivas viene determinado por el valor max-repetitions. Además podemos pedir variables sin repetir, cuyo número viene determinado por el valor non-repeaters. De este modo podemos pedir más información en una cantidad menor de mensajes, con lo que aumentamos la efectividad. Este comando se basa en realizar continuas operaciones GetNext hasta obtener la cantidad de datos a devolver.
  - *SetRequestPdu:* mensaje que modifica una variable de la MIB. Si ha habido algún error a la hora de modificar, el agente lo comunicará en el mensaje de respuesta mediante el código de error correspondiente.
- Mensajes enviados por el agente al gestor:
  - *GetResponsePdu:* mensaje de respuesta para los mensajes que nos envía el gestor. En él se incluyen los valores de las variables requeridas, y el código del error en caso de que ocurra alguno.
  - *TrapPdu:* mensaje generado y transmitido de forma asíncrona como respuesta a un evento excepcional. En él se incluye el sysUpTime, que es el tiempo que lleva encendido el dispositivo, además de las variables que correspondan al tipo de trap generado.
- Mensajes enviados de gestor a gestor:
  - *InformationRequestPdu:* mensaje para enviar una alerta de un gestor a otro, introducido a partir de la versión 2.

La Figura 2.1 ilustra las operaciones a partir de SNMPv2c (ya que incluye el GetBulk), con la dirección respectiva: del gestor al agente o viceversa.

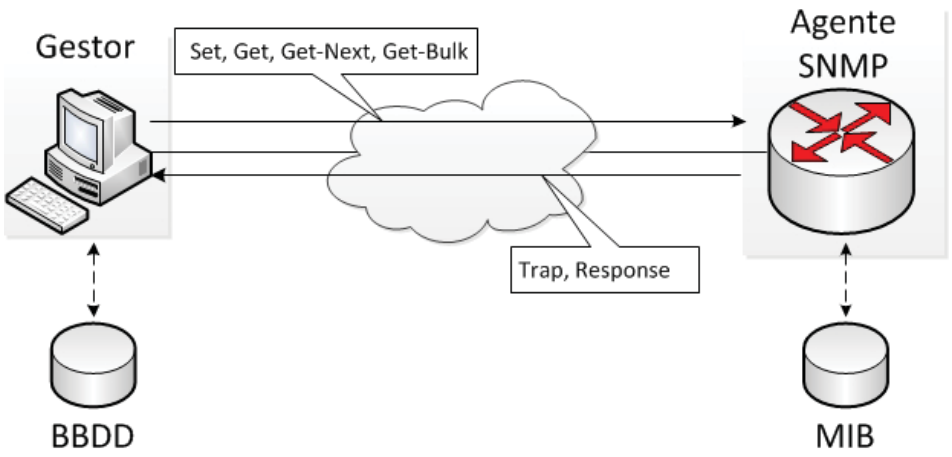

Figura 2.1: Comunicación SNMP entre agente y gestor.

2.1.3 Bases de datos en SNMP: MIB

Una de las características más importantes de la arquitectura SNMP es la MIB. La MIB es una BBDD virtual que organiza jerárquicamente en forma de árbol la información usada para gestionar las entidades en un entorno de red. Cada objeto gestionado se identifica unívocamente mediante su OID. La filosofía de este lenguaje de programación de BBDD es favorecer la simplicidad y extensibilidad de las MIB, permitiendo sólo estructuras simples de datos. Cada objeto dentro de la MIB está formado por una variable, que es el valor del objeto, y su identificador (OID). El OID está formado por una secuencia de números y puntos, representando cada uno un salto de nivel dentro del árbol que compone la MIB.

Las MIBs se dividen en dos tipos, públicas y privadas. Las públicas están definidas mediante estándares y proporcionan información general del sistema. Las privadas están definidas por los fabricantes y ofrecen información más detallada y concreta. La MIB más conocida es la 'mib-2' [17], dado que tiene una amplia información tanto de la red, por ejemplo parámetros estadísticos de tráfico, como de dispositivos, por ejemplo el uso de CPU o de memoria, aunque no contiene información de más alto nivel, referente a aplicaciones o al sistema operativo.

Una gran ventaja de este protocolo es que, como ya hemos comentado, al estar las MIBs estandarizadas, y al haber una gran cantidad definidas de ellas, implementando lo que nosotros queramos en cada agente recogeremos la información que necesitemos.

Una MIB ha de escribirse cumpliendo una serie de reglas siguiendo un lenguaje formal y ha de estar compuesta por Grupos, Objetos o Tablas. Este lenguaje se llama SMI (2.2.1). Un Grupo es un nodo donde cuelgan los Objetos y las Tablas. Un Objeto es una variable de un Grupo que contiene información. Una Tabla es un conjunto de variables (Objetos de Tabla) que dependen de un índice llamado instancia. En la Figura 2.2 podemos ver un ejemplo de la estructura en árbol de una MIB.

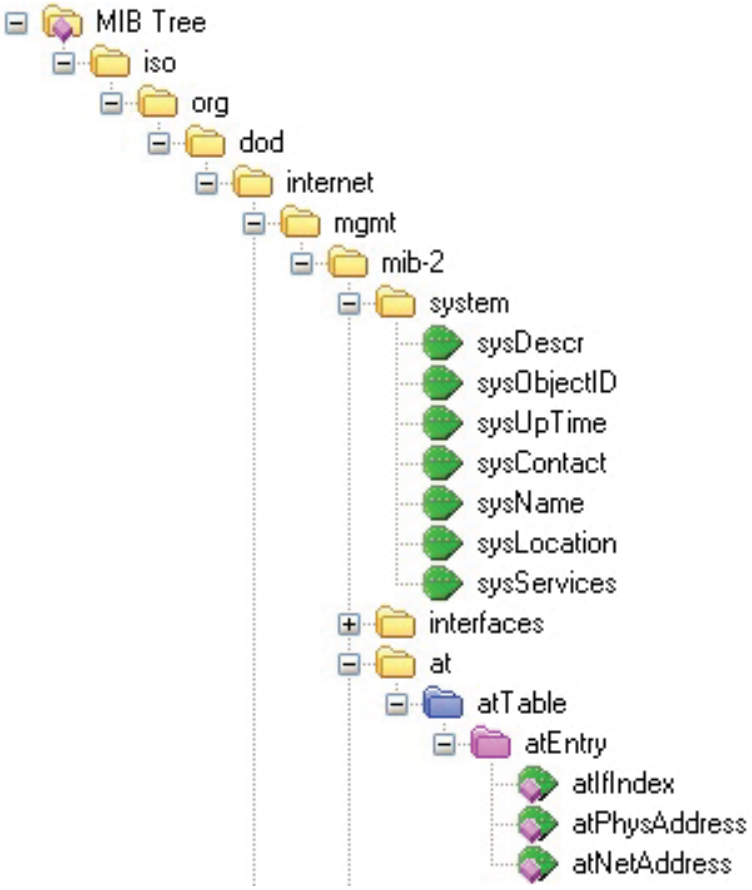

Figura 2.2: Estructura en árbol de una MIB.

2.1.4 Seguridad en SNMP: SNMPv3

Como hemos introducido previamente, la versión 3 (SNMPv3) [18] fue desarrollada para mejorar la seguridad. Nos permite control de acceso, confidencialidad y privacidad en las comunicaciones a través de la autenticación del usuario y del cifrado de la comunicación.

Para la definición y aplicación de políticas de seguridad se utilizan dos bloques dentro de la propia entidad SNMP, llamados USM (User-based Security Model) y VACM (Viewbased Access Control Model). USM se encarga de la gestión de usuarios e información necesaria para el cifrado y seguridad de las comunicaciones. Para ello utiliza funciones resumen del cuerpo del mensaje (MD5 o SHA-1) para evitar la modificación del mensaje por parte de un tercero o ataques de suplantación de identidad. Para asegurar la confidencialidad SNMPv3 se apoya en algoritmos de cifrado (DES o AES de 128 bits). VACM gestiona los privilegios que cada usuario USM posee y las Tablas y Objetos a los que puede acceder. Para ello asigna a cada usuario una pareja de valores «securityModel, securityName» con los que determina el nivel de seguridad aplicado a cada usuario. En la Figura 2.3 podemos observar cómo se lleva a cabo este proceso de autenticación y cifrado.

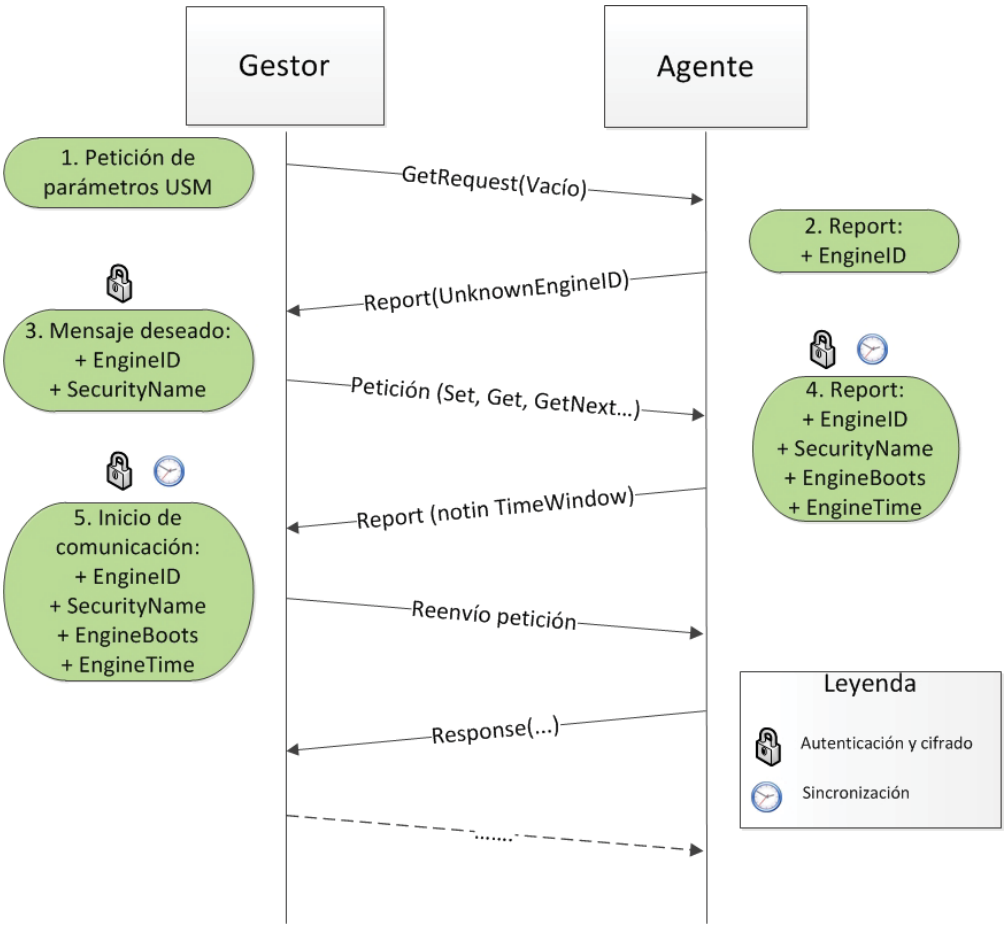

Figura 2.3: Inicio comunicación SNMPv3 (Caso AuthPriv).

El agente y el gestor necesitan sincronizarse antes de comenzar la comunicación. Para realizar la sincronización, el gestor copia los valores de snmpEngineTime (indica el tiempo que ha transcurrido desde la última vez que se reinició por última vez), snmpEngineBoots (parámetro que indica cuántas veces se ha reiniciado ese agente) y latestReceivedEngineTime del agente. Para mantener la sincronización, el agente manda sus parámetros temporales junto con su snmpEngineID en cada mensaje, que es un identificador único de cada entidad SNMP, y si esos valores son correctos el gestor actualiza los valores.

Se establece una ventana en la cual los mensajes recibidos no se consideran con retardo y son válidos. Hay que tener cuidado de no poner una ventana demasiado pequeña o grande. En caso de que el timeout expire, el mensaje se considera inválido, se devuelve un error de notInTimeWindow y se espera la retransmisión del mensaje.

SNMPv3 asegura tanto modelos de seguridad como niveles de seguridad. Una combinación de un modelo de seguridad y un nivel de seguridad determina que mecanismo de seguridad es empleado en el manejo de un paquete SNMP. La Tabla 2.1 identifica lo que las combinaciones de modelos de seguridad y nivel significan.

Modelo	Nivel	Autenticación	Cifrado	¿Qué ocurre?
v1	noAuthNoPriv	Nombre comunidad	No	Usa el nombre de la comunidad para la autenticación
v2c	noAuthNoPriv	Nombre comunidad	No	Usa el nombre de la comunidad para la autenticación
v3	noAuthNoPriv	Nombre de usuario	No	Usa el nombre de usuario para la autenticación
v3	AuthNoPriv	MD5 o SHA	No	Proporciona autenticación basada en los algoritmos HMAC-MD5 o HMAC-SHA
v3	AuthPriv	MD5 o SHA	DES o AES	Proporciona autenticación basada en los algoritmos HMAC-MD5 o HMAC-SHA. También proporciona cifrado DES o AES

Tabla 2.1: Combinación de los modelos de seguridad.

2.2 El lenguaje SMI

El lenguaje SMI (Storage Magament Initiative) fue creado por SNIA [19] (Storage Networking Industry Association) para desarrollar y estandarizar tecnologías interoperables de gestión de almacenamiento y además, promover el almacenamiento entre redes y comunidades de usuarios finales. Esta iniciativa está soportada por muchos grupos dentro de la organización SNIA como: SMF [20] (Storage Management Forum), TWG [21] (SNIA‘s Tacnicall Working Groups) y CTP [22] (Conformance Test Programs).

El lenguaje SMI puede desarrollar las actividades de SNIA a las siguientes áreas:

- Desarrollo tecnológico.
- Pruebas de calidad de productos.
- Comercialización, educación y formación.

La estructura de almacenamiento de la información en SMI está definida usando ASN.1. Este lenguaje define a su vez las reglas para describir el almacenamiento de la información (MIB).

El lenguaje SMI define el marco en el cual un módulo MIB puede ser definido o construido. En otras palabras, esto define los componentes de un módulo MIB y el lenguaje formal para describir los objetos manejados. El lenguaje SMI especifica que todos los objetos empleados deben tener un nombre específico y una sintaxis definida. El nombre se caracteriza por el Object Identifier (OID) único. La sintaxis define el tipo de datos de un Objeto (por ejemplo, integer o string).

2.2.1 Evolución del lenguaje SMI

Actualmente está en vigor el uso del lenguaje SMIv2 definido en la RFC 2578 [23]. Esta versión del lenguaje surge como evolución lógica del primer estándar SMIv1. La principal mejora que se encuentra en SMIv2 es que se dotó de una mayor cantidad de tipos de datos, y ofrece una sintaxis más rica y exacta para definir módulos MIB. Además SMIv2 contiene a SMIv1.

Hoy en día, la mayoría de dispositivos y aplicaciones se implementan usando las reglas SMIv2, pero todavía queda algún dispositivo que requiere SMIv1, por lo que se necesita de la presencia de ambos.

Cabe destacar que los módulos de MIB que usan los datos específicos SMIv2 como estructura de datos, no requiere SNMPv2c como protocolo de transporte (se puede usar SNMPv1). SNMPv2c se requiere solo en caso de un módulo MIB definido con SMIv2 contenga un tipo de datos no soportado por SMIv1 (por ejemplo Counter64), porque SNMPv1 no soporta este tipo de datos. Si el módulo MIB no contiene ningún tipo de datos desconocidos por SMIv1, los compiladores MIB pueden traducir módulos MIB de SMIv2 a SMIv1.

Los tipos de datos definidos para SMIv1 son los mostrados en la Tabla 2.2:

Tipo Dato	**Descripción**
INTEGER	Números tanto positivos como negativos
OCTET STRING	Usado para transmitir datos binarios, pero la transmisión es en múltiplos de 8 bits
OBJECT IDENTIFIER	Identificación sin ambigüedad de una entidad registrada en la MIB
Gauge	Número entero no negativo que puede aumentar o disminuir, pero que llega a un valor máximo
Counter	Número entero no negativo que se incrementa en +1 hasta que esto alcance un valor máximo, que se reinicializa a 0
Time Ticks	Centésimas de segundo desde un acontecimiento
IpAddress	Codificación de 4 octetos para la Dirección IP
NetworkAddress	Codificación de 4 octetos para la Dirección IP de la red
Opaque	Método arbitrario donde un dato puede ser pasado por OCTET STRING

Tabla 2.2: Tipos de datos SMIv1.

Los tipos de datos definidos para SMIv2, a parte de *INTEGER, OCTET STRING, OBJECT IDENTIFIER, Time Ticks, IpAddress y Opaque* que se mantienen de SMIv1, son los mostrados en la Tabla 2.3:

Tipo Dato	**Descripción**
Integer32	Igual al tipo de dato INTEGER
Unsigned32	Representa valores decimales comprendidos entre 0 y 2^{32}- 1
Gauge32	Igual al tipo de dato Gauge
Counter32	Igual al tipo de dato Counter
Counter64	Similar a Counter32 pero en este caso el máximo valor es 2^{64}- 1
BITS	Permite enumerar enteros no negativos llamados *bits*

Tabla 2.3: Tipos de datos SMIv2.

2.2.2 Estructura de una MIB con SMI

La estructura que tiene una MIB se divide en varias partes:

- Lo primero de todo es el comienzo de la definición de la MIB con su nombre y la sintaxis clave: 'DEFINITION ::= BEGIN'
  - nombre *DEFINITION ::= BEGIN*
- A continuación puede estar definida (es opcional) una zona para importar tipos de datos o nodos. Para definir esta zona, se comienza por la palabra clave 'IMPORT' y a continuación lo que se importa y de donde se importa:
  - *IMPORTS* internet, mgmt *FROM* RFC1155-SMI
- Otro bloque opcional que puede contener una MIB es el llamado definición del módulo de identidad. Aquí se define la información asociada al creador de la MIB, información de contacto y fecha de la publicación o actualización de la MIB:
  - *mibDesign MODULE-IDENTITY*
  - *LAST-UPDATED* 'Fecha y hora de la última actualización'
  - *ORGANIZATION* 'Organismo que ha desarrollado el módulo MIB'
  - *CONTACT-INFO* 'Información de contacto'
  - *DESCRIPTION* 'Descripción del Module-Identity'
  - *::=* {nodoPredecesor numeroAsignado}
- Posteriormente es el turno de la definición de los nodos (o grupos). Se pone el nombre del nodo a definir, la sintaxis de definición '::=' y se escribe entre llaves '{ }' el nombre del nodo predecesor seguido del último nodo del OID asignado:
  - mib-2 *::=* { mgmt 1 } (mib-2 hereda el OID de mgmt más un 1 (1.3.6.1.2.1))
- Una vez se hayan definido todos los nodos que componen la MIB, y toda la información adicional opcional, se definen sus Objetos:
  - Empieza con el nombre del Objeto seguido de la sintaxis '*OBJECT-TYPE*'.
  - A continuación se especifica la sintaxis del Objeto (ejemplo en la Figura 2.4), para ello se pone '*SYNTAX*' seguido del tipo de dato que sea el Objeto, puede ser cualquier tipo de dato del lenguaje SMI, cualquier otro importado o puede definirse una Tabla ('*SYNTAX SEQUENCE OF* nombreTabla', ejemplo en la Figura 2.5) o el contenido de la Tabla (*SYNTAX* nombreTabla, ejemplo en la Figura 2.6).

- Se sigue definiendo el tipo de acceso del Objeto mediante la sintaxis '*MAX-ACCESS*' o '*ACCESS*' seguido del tipo de acceso ('not-accessible', 'accessible-for-notify', 'read-only', 'read-write', 'read-create').
- A continuación se pone el estado del Objeto poniendo '*STATUS*' seguido del estado (current, obsolete, deprecated).
- Sigue con la zona de descripción: '*DESCRIPTION*' seguido de la definición del Objeto.
- Si el Objeto definido es el contenido de una Tabla ha de estar definida la sintaxis '*INDEX*' indicando los índices de la Tabla entre llaves.
- Por último ha de ponerse el nodo de donde cuelga el Objeto '::={mib-2 2}'.

```
sysUpTime OBJECT-TYPE
    SYNTAX  TimeTicks
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The time (in hundredths of a second) since the
            network management portion of the system was last
            re-initialized."
    ::= { system 3 }
```

Figura 2.4: Definición del Objeto *sysUpTime*.

```
atTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF AtEntry
    ACCESS  not-accessible
    STATUS  deprecated
    DESCRIPTION
            "The Address Translation tables contain the
            NetworkAddress to `physical' address equivalences.
            Some interfaces do not use translation tables for
            determining address equivalences (e.g., DDN-X.25
            has an algorithmic method); if all interfaces are
            of this type, then the Address Translation table
            is empty, i.e., has zero entries."
    ::= { at 1 }
```

Figura 2.5: Definición de la Tabla *atTable*.

```
atEntry OBJECT-TYPE
    SYNTAX  AtEntry
    ACCESS  not-accessible
    STATUS  deprecated
    DESCRIPTION
            "Each entry contains one NetworkAddress to
            `physical' address equivalence."
    INDEX   { atIfIndex,
              atNetAddress }
    ::= { atTable 1 }
```

Figura 2.6: Definición de los campos de la Tabla *atTable*.

- Una MIB también puede tener definida una zona de notificación para poder mandar información al gestor en caso de que se programe algún evento para tal fin:
  - Comienza con el nombre que se le da a la notificación seguido de la sintaxis '*NOTIFICATION-TYPE*'.
  - Ha de estar el campo '*STATUS*' seguido del estado (current, obsolete, deprecated).
  - Sigue con la zona de descripción: '*DESCRIPTION*' seguido de la descripción de este campo.
  - Ha de finalizar poniendo el nodo de donde cuelga el Objeto '::={mib-2 3}'.

Capítulo 3

Desarrollo Tecnológico: Bloques implementados

3.1 Descripción general

El sistema se compone de diferentes bloques:

- MIB Builder: El objetivo del MIB Builder es ayudar a crear la definición formal de la MIB a partir de un interfaz gráfico. Es la parte previa al paso de crear la BBDD de la MIB y consta de una parte dedicada a la creación del esquema en forma de árbol de la MIB donde se le pueden añadir una serie de objetos determinados en otra parte del panel. El componente añadido al esquema tendrá una serie de parámetros, los cuales se podrán editar en la zona central del panel. Una vez que hemos terminado de diseñar el árbol de nuestra MIB, un programa desarrollado para tal fin convertirá el árbol dibujado de la MIB en un fichero de texto con su definición formal (en el lenguaje SMI).

- BBDD Creation: El objetivo del BBDD Creation es crear una serie de tablas en la BBDD a partir de un fichero de texto en el que se encuentra la definición formal de la MIB. Para ello se ha desarrollado un programa que permite leer ese fichero de texto. Además se ha desarrollado un algoritmo de conversión entre los elementos del lenguaje SMI y los módulos de la BBDD, una estructura genérica de la misma y un posterior algoritmo de lectura para facilitar la interacción del agente con su MIB. Se trata de reconocer las partes relevantes y fundamentales para la elaboración de las tablas (cuya estructura se propone después de hacer un estudio sobre la materia en cuestión) que componen la BBDD. Una vez que se reconoce la información deseada, se va almacenando en unas variables dependientes de una estructura de clases Java también diseñadas para tal fin. Con la información de control que proporciona el usuario en el interfaz sobre la BBDD y estas variables que contienen la información de las tablas de la BBDD, se genera dicha BBDD

que dará soporte a la MIB.

- Agente SNMP: Su objetivo es establecer comunicaciones SNMP con el gestor externo y poner a su disposición información almacenada dentro de la MIB que implementa. Consta de dos partes claramente diferenciadas, por una parte está una zona dedicada a gestionar las cuentas de usuarios que tendrán acceso al agente, en la cual se pueden añadir usuarios (definir que permisos tienen), mostrar los ya existentes o eliminarlos. Por otra parte está la zona de puesta en marcha del agente que consta de campos donde se puede introducir el puerto y la dirección IP que va a tener.

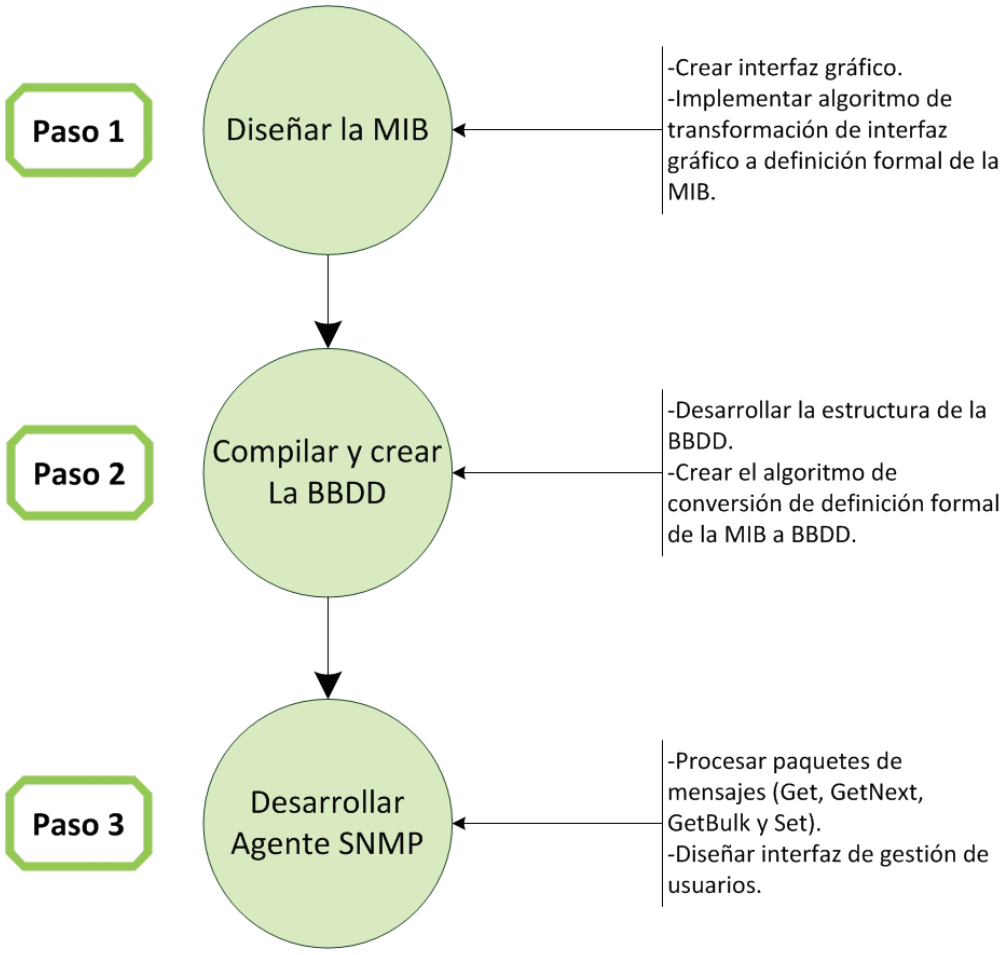

Figura 3.1: Workflow del problema a resolver.

Cualquier gestor que quiera conectarse con el agente generado por nuestro sistema, lo hará a través de Internet mediante el protocolo SNMP. Pero para que dicho agente esté implementado y puesto en marcha, debe seguirse el siguiente proceso a través de nuestro programa: se ha de diseñar la estructura de la MIB en nuestro *MIB Builder* para obtener la definición formal de la MIB (o se puede tener ya escrita). A continuación, debe introducirse el fichero de texto con la definición formal de la MIB en el siguiente módulo,

el *BBDD Creation*, junto con los datos de control de la BBDD para poder conectarnos a ella. Finalmente se deberá pasar al interfaz *Agente SNMP* , introducir los usuarios (con sus respectivos permisos) que podrán tener acceso, y ponerlo en marcha dándole al botón correspondiente. La Figura 3.1 trata de mostrar el proceso seguido para resolver el problema.

3.2 MIB Builder

3.2.1 Diseño

El paso más importante para la resolución del objetivo de crear el Mib Builder es la creación del interfaz con el que interactúe el usuario. El criterio de diseño que se ha tenido en cuenta es que cualquier usuario debe ser capaz de utilizar el programa de una manera fácil e intuitiva.

Como se puede ver el la Figura 3.2 el interfaz está dividido en tres zonas:

- La zona 1 es la dedicada al diseño del árbol de la MIB.
- La zona 2 se compone de una serie de objetos para la edición y diseño de la MIB.
- La zona 3 es la dedicada para nombrar la MIB y poner la ubicación del archivo de texto salida.

Figura 3.2: Zonas del Interfaz *MIB Builder*.

En la zona destinada para crear el diseño del árbol de la MIB, la zona 1, se ha puesto una tabla inicialmente vacía para permitir al usuario crear la estructura en árbol de una

manera sencilla. El anexo B contiene una guía de usuario en el que se detallan los pasos a seguir para la creación de una nueva MIB.

En la zona 2 es donde hay que introducir los campos de cada componente cumpliendo con la sintaxis del lenguaje SMI. En la Figura 3.3 se puede ver la correlación existente entre los campos de la zona de edición del Mib Builder y la definición formal de la MIB a través del ejemplo de la definición del Objeto *sysUpTime* del grupo *System* de la *mib-2*.

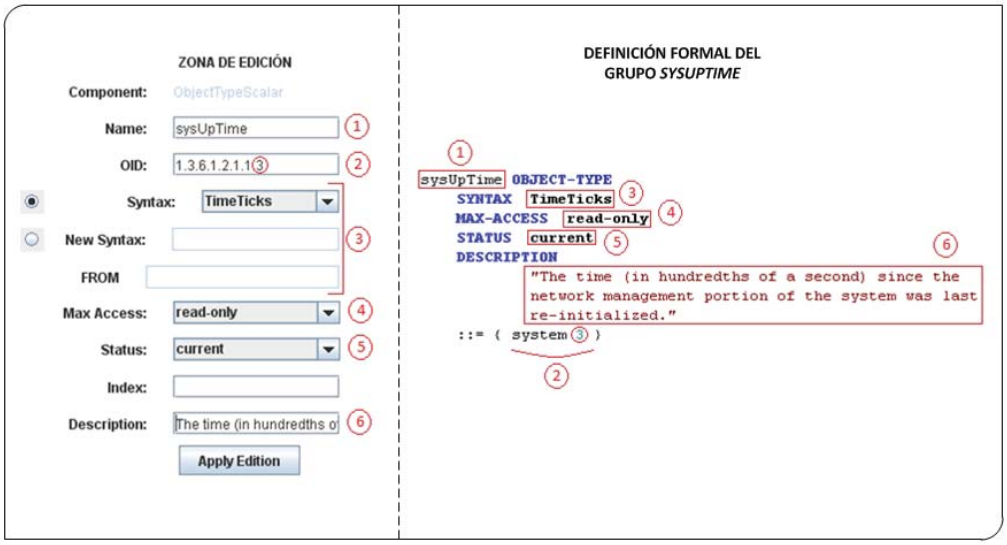

Figura 3.3: Definición del grupo *sysUpTime*.

En cuanto a la zona donde se definen las características de los objetos introducidos en el árbol de la MIB se tienen los siguientes campos:

- En el campo *Componet* se pondrá automáticamente el tipo de componente que se haya introducido en el esquemático de la MIB. Es un tipo clave para poder identificar el tipo de componente ya que así su posterior procesamiento será más sencillo debido a que cada tipo de componente tiene unas características semánticas específicas en la definición formal de la MIB.

- En el campo *Name* aparecerá un nombre por defecto, dependiendo del tipo de componente del que se trate, pero que el usuario podrá modificar.

- El campo *OID* se va completando automáticamente partiendo del OID del grupo anterior, y el usuario podrá modificar el último trozo correspondiente al componente introducido si desea que sea un valor concreto.

- Después viene la zona donde se ha de especificar la *sintaxis* del componente que sea necesario (todos menos OBJECT IDENTIFIER). Se tienen dos opciones:

  – Que la sintaxis sea de un tipo de dato predefinido en cuyo caso se

seleccionará esa opción (seleccionando el botón de su izquierda) y se elegirá uno de esos tipos de datos.

– Que el tipo de dato sea distinto de uno definido o para la creación de una Tabla, en cuyo caso se ha de hacer uso de los campos *New Syntax* y *FROM* para definirlo.

- Para rellenar el campo *Max-Access* y *Status* se ha de seleccionar una de las opciones disponibles.
- El campo *Index* solo hay que rellenarlo en el caso de definir una Tabla para indicarle el índice o los índices que posea dicha Tabla.
- En el campo *Description* se pone la descripción que se desee decir para expilcar las características del componente.

Esta información introducida se puede editar en cualquier momento que se precise, antes de crear finalmente el archivo de texto de la MIB.

En la última zona, la zona 3, se debe introducir el nombre de la MIB y la ubicación de la carpeta de destino donde se quiera generar el archivo de texto de la definición formal de la MIB, para permitir al usuario elegir la carpeta contenedora del archivo.

3.2.2 Implementación

Para la implementación de este bloque ha sido necesario crear varias clases Java además de su interfaz, mostrado en la Figura 3.4.

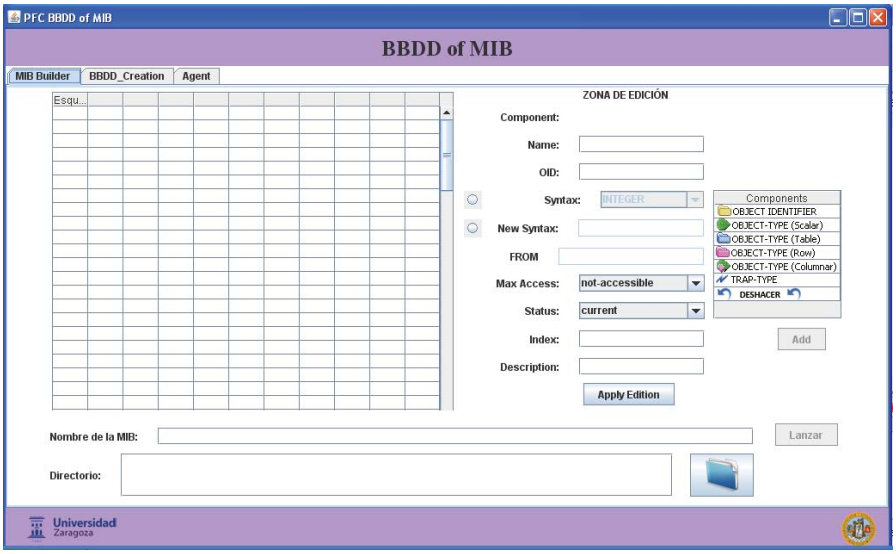

Figura 3.4: Interfaz MIB Builder.

La primera clase Java que fue necesario crear se trata de la clase *component*, desarrollada especialmente para almacenar toda la información introducida en el interfaz

con respecto al componente introducido en la zona de diseño del árbol. Esta clase contiene todos los campos de los que se pueden introducir información de cualquier componente.

El árbol MIB es en definitiva un conjunto de componentes, por lo que se trata a efectos prácticos del programa de un array de la clase *component*.

Para que la información de cada componente se haga efectiva es necesario pulsar el botón 'Apply'. Esto es así porque al introducir un nuevo componente al árbol de la MIB, se genera automáticamente un nuevo elemento en el array que contiene los componentes del árbol. De esta manera, al pulsar el botón 'Apply' le indicamos al programa que hemos completado o modificado la información de los campos que se genera automáticamente al añadir el componente en el árbol y que la información que presentaba anteriormente ha quedado obsoleta.

De igual manera, para modificar la información de cualquier componente ya introducido en el árbol, se han implementado varias funciones Java para obtener la información del array de componentes introducidos y poder modificarla.

3.3 BBDD Creation

3.3.1 Algoritmo de conversión

Este bloque se encarga en primer lugar de leer la definición formal de la MIB y extraer de ella los componentes de la MIB. El fichero de texto que contiene la MIB se lee hasta detectar el comienzo, ya que suele ir precedida de una descripción hecha por el autor sobre dicha MIB. Una vez reconocido el inicio de la MIB ('DEFINITION::=BEGIN') hemos de detectar todos los componentes existentes y los nodos para almacenarlos en un conjunto de variables y clases Java para poder automatizar el proceso de crear la BBDD. En la Figura 3.5 se muestra un resumen del proceso que realiza el algoritmo.

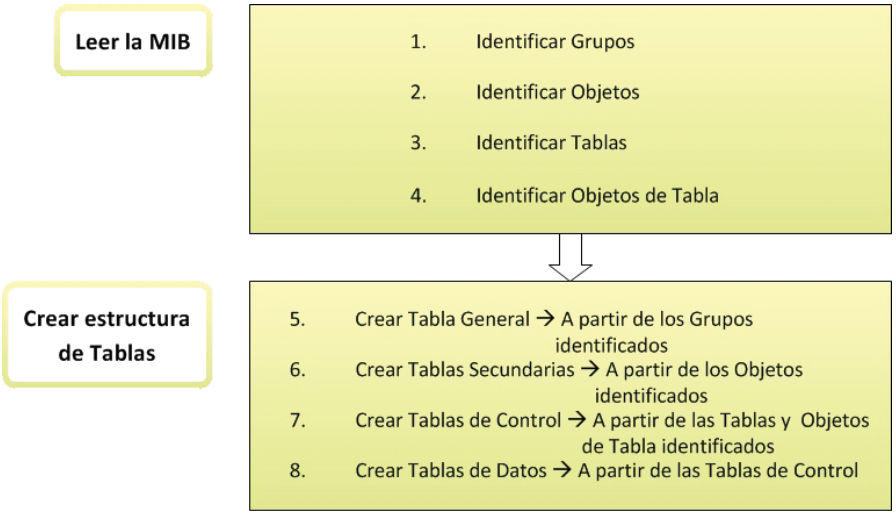

Figura 3.5: Funciones del algoritmo de conversión.

Lo primero de todo es almacenar la información de los grupos existentes en la MIB, ya que son los primeros en ser descritos en la definición formal de cualquier MIB. Puesto que se conoce el OID completo de los Grupos *mgmt* e *internet* (así como todos los Grupos anteriores a éstos), se almacena en un conjunto de variables específicamente diseñado para este fin (como si de una tabla se tratase) el nombre del Grupo y su OID completo. Cada vez que se reconoce un nuevo Grupo en la definición formal de la MIB se actualiza este conjunto de variables añadiendo el nuevo Grupo reconocido con su OID, compuesto por el OID de un Grupo que ya tenemos más el nuevo número introducido.

A continuación se reconocen los Objetos, las Tablas y los Objetos de las Tablas de la MIB. Los Objetos de la MIB se reconocen porque se definen con la sintaxis 'OBJECT-TYPE'. Las Tablas porque la definición del campo SYNTAX se define como 'SEQUENCE OF' *nombreTabla*. Finalmente, los Objetos de Tabla son iguales que los Objetos y los identificamos porque en su OID el nodo precedente es el nombre de la Tabla. Pero no se trata solamente de reconocer estos campos, sino que también hay que almacenar toda esa información.

En la Figura 3.6 se muestra la sintaxis clave detectada en la MIB y las partes almacenadas.

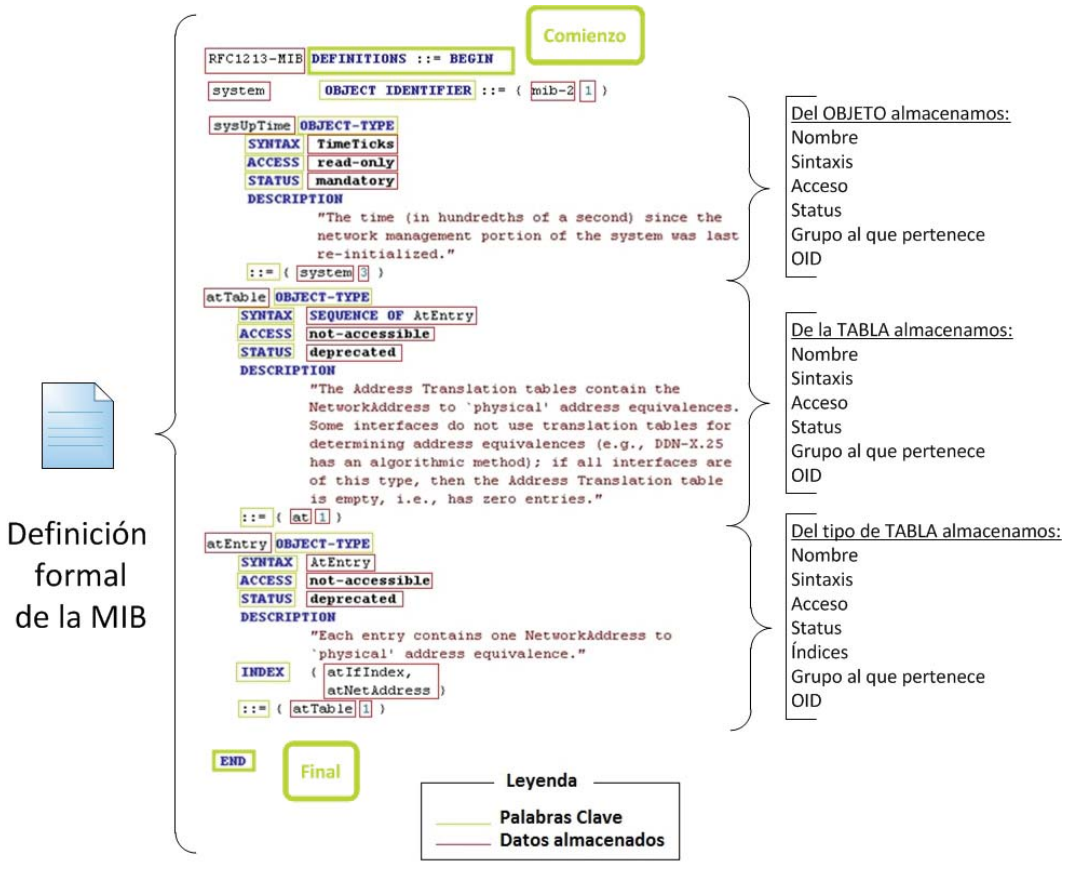

Figura 3.6: Partes a almacenar de la MIB.

Para realizar la organización de la BBDD en la que se implementan las MIBs se propone un esquema basado en la utilización de distintos tipos de tablas genéricas. El diseño propuesto permitirá mantener la estructura de árbol de la MIB y permitir así la implementación de cualquier MIB. La estructura genérica propuesta se basa en la utilización de cuatro tipos de tablas para almacenar los Objetos de una MIB genérica de forma ordenada: 'Tabla General', 'Tabla Secundaria', 'Tabla de Control' y 'Tabla de Datos', cuya estructura de tablas se describe en el anexo C, las cuales sirven para almacenar toda la información relevante de los Grupos, Objetos, Tablas y Objetos de Tablas presentes en la MIB.

Las variables *TablaGeneral* y *TablaSecundaria* tienen la misma estructura de tabla (C.1) y en ellas se almacenan toda la información de los Grupos y Objetos que pertenezcan a un Grupo y no sean Objetos de Tabla (nombre, OID, acceso, tipo de dato, valor...). En la *TablaGeneral* tendremos todos los nodos y Objetos que cuelgan de nuestro nodo raíz, y en la *TablaSecundaria* tendremos todos los Objetos y Grupos en ella definidos. Sólo puede haber una *TablaGeneral*, en cambio, pueden existir varias *TablasSecundarias* (tantas como Grupos haya definidos por debajo del nodo raíz).

La variable *TablaControl* contendrá todos los Objetos de Tabla de una Tabla definida en la MIB (nombre, OID, índice de la tabla, tamaño de la tabla...). Y la variable *TablaDatos* contendrá todas las instancias de la Tabla.

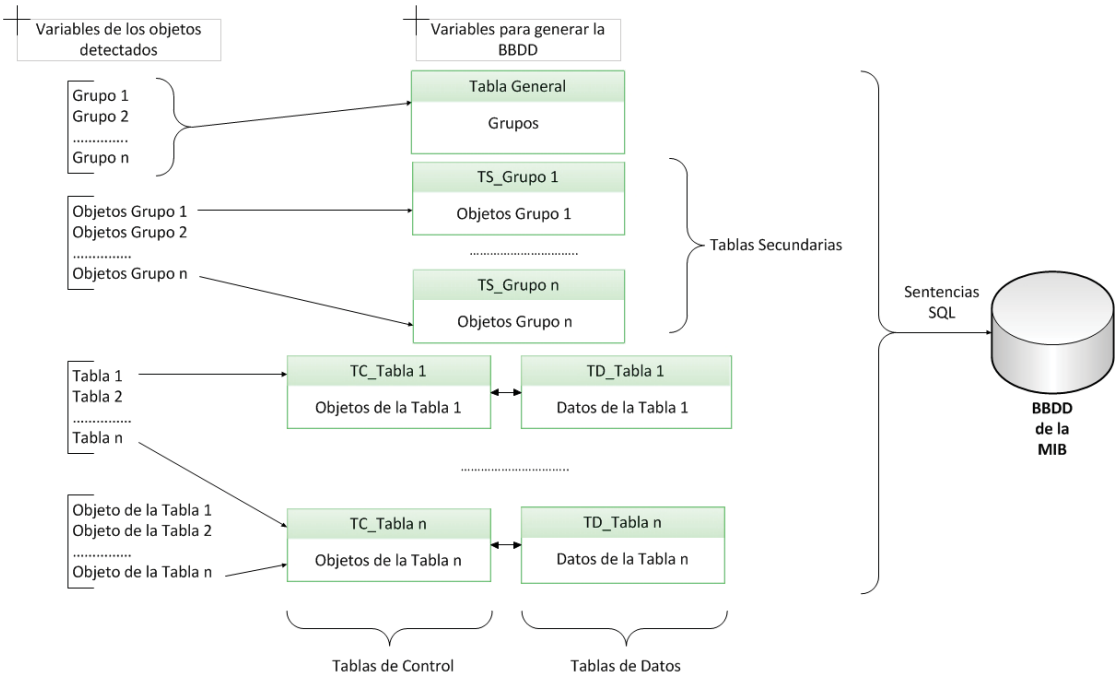

Figura 3.7: Relación entre compontes y tablas creadas.

En la Figura 3.7 se puede ver como se forman las tablas con los Objetos almacenados (La *Tabla General* a partir de los grupos, las *Tablas Secundarias* una por grupo que

contien los Objetos de cada Grupo, una *Tabla de Control* por cada Tabla de la MIB que contienen todos los Objetos de cada Tabla y una *Tabla de Datos* por cada *Tabla de Control* donde se almacenarán los datos de una Tabla).

Para llegar a la propuesta de estas tablas como estructura genérica de la BBDD, se ha hecho un estudio de los Grupos *system* e *interfaces* de la MIB 'mib-2' y del Grupo *alarm* de la MIB 'RMON' [24] ya que estas dos MIBs engloban todos los casos posibles de diferentes MIBs (Grupos con únicamente Objetos, Grupos con únicamente Tablas, Grupos con Objetos y Tablas, Grupos con Tablas que poseen permisos de lectura y escritura especiales...).

3.3.2 Implementación

El programa diseñado para crear la BBDD a partir de la definición formal de una MIB, hace uso de estas clases Java para almacenar la información de los componentes de la MIB. Dicho programa realiza una lectura del archivo de texto y detecta la sintaxis característica para discriminar un tipo de componente de otro, para de esta manera, almacenar la información. Una vez que el programa haya leído toda la información del fichero de texto, se estará en disposición de implementar la BBDD ya que se tendrá toda la información de los componentes de la MIB almacenada en variables.

Para implementar la BBDD una vez que se posean todos los datos que debe contener se hace uso de las variables del mismo tipo de las tablas genéricas definidas anteriormente (anexo C): *TablaGeneral, TablaSecundaria (TS), TablaControl (TC)* y *TablaDatos (TD).*

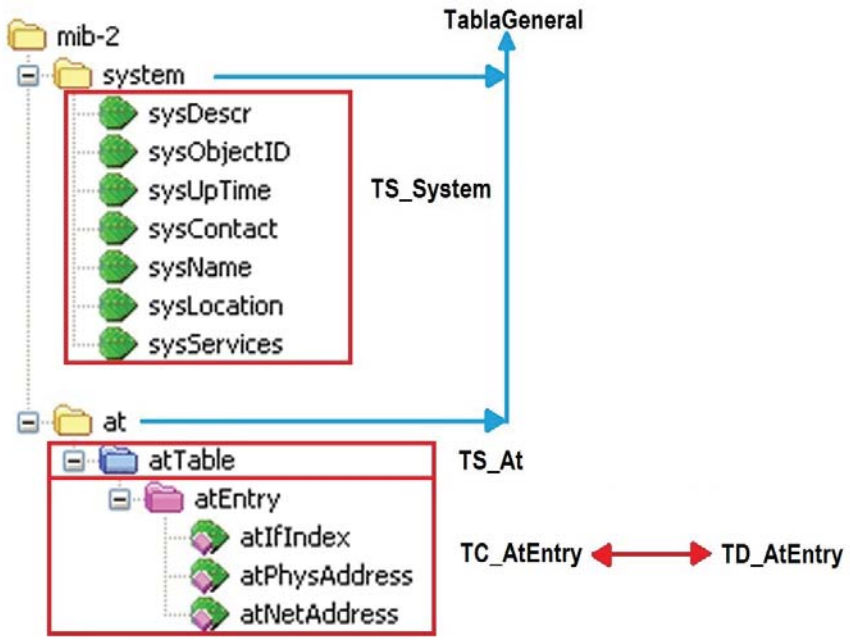

Figura 3.8: Relación entre definición formal y tablas creadas.

En la Figura 3.8 se presenta un ejemplo de las tablas creadas de una MIB compuesta por los grupos *system* y *at* de la mib-2.

Las tablas generadas al pasar por el algoritmo creado, mostradas en la Figura 3.9, son las siguientes:

- *Tabla General*.
- *Tabla Secundaria* del Grupo system (TS_System).
- *Tabla Secundaria* del Grupo at (TS_At).
- *Tabla de Control* de la Tabla atEntry (TC_AtEntry).
- *Tabla de Datos* (TD_AtEntry) asociada a la *Tabla de Control* TC_AtEntry.

TablaGeneral

OID	ORDEN	NAME	NEXT_TABLE	TYPE_VALUE	VALUE	ACCESS	STATUS	NEXT_OID
1.3.6.1.2.1.1	1	system	ts_system	0	NONE	0	stats	1.3.6.1.2.1.1.1.0
1.3.6.1.2.1.3	2	at	ts_at	0	NONE	0	stats	nextOID

TS_System

OID	ORDEN	NAME	NEXT_TABLE	TYPE_VALUE	VALUE	ACCESS	STATUS	NEXT_OID
1.3.6.1.2.1.1.1	1	sysDescr	nextTABLE	DisplayString	NONE	2	mandatory	1.3.6.1.2.1.1.2.0
1.3.6.1.2.1.1.2	2	sysObjectID	nextTABLE	OBJECT IDENTIFIER	NONE	2	mandatory	1.3.6.1.2.1.1.3.0
1.3.6.1.2.1.1.3	3	sysUpTime	nextTABLE	TimeTicks	NONE	2	mandatory	1.3.6.1.2.1.1.4.0
1.3.6.1.2.1.1.4	4	sysContact	nextTABLE	DisplayString	NONE	3	mandatory	1.3.6.1.2.1.1.5.0
1.3.6.1.2.1.1.5	5	sysName	nextTABLE	DisplayString	NONE	3	mandatory	1.3.6.1.2.1.1.6.0
1.3.6.1.2.1.1.6	6	sysLocation	nextTABLE	DisplayString	NONE	3	mandatory	1.3.6.1.2.1.1.7.0
1.3.6.1.2.1.1.7	7	sysServices	nextTABLE	INTEGER	NONE	2	mandatory	nextOID

TS_At

OID	ORDEN	NAME	NEXT_TABLE	TYPE_VALUE	VALUE	ACCESS	STATUS	NEXT_OID
1.3.6.1.2.1.3.1	1	atTable	tc_atEntry	SEQUENCE OF AtEntry	NONE	0	deprecated	nextOID

TC_AtEntry

OID	ORDEN	NAME	NEXT_TABLE	INDICES	TYPE_VALUE	VALUE	ACCESS	STATUS	NEXT_OID
1.3.6.1.2.1.3.1.1.1	1	atIfIndex	td_atEntry	atIfIndex, atNetAddress	INTEGER	0	3	deprecated	nextOID
1.3.6.1.2.1.3.1.1.2	2	atPhysAddress	td_atEntry	atIfIndex, atNetAddress	PhysAddress	0	3	deprecated	nextOID
1.3.6.1.2.1.3.1.1.3	3	atNetAddress	td_atEntry	atIfIndex, atNetAddress	NetworkAddress	0	3	deprecated	nextOID

TD_AtEntry

atIfIndex	atPhysAddress	atNetAddress

Figura 3.9: Tablas creadas para la MIB propuesta.

El interfaz implementado (ver Figura 3.10) está dividido en cuatro zonas:

- La zona 1 (*Archivo definición MIB*) permite introducir el nombre del archivo de texto con la definición formal de la MIB. Se puede escribir la dirección completa, o ayudarse del botón de al lado para seleccionar el archivo.
- En la zona 2 (*BBDD*) se recogen los elementos relacionados con la creación de la BBDD en MySQL para poder acceder a ella, el nombre de la BBDD a crear y el nombre del fichero de texto que se quiera dar al fichero de información que se genera en este apartado. Hemos de darle al botón 'Apply' para fijar los valores introducidos. El fichero de información contendrá la información más relevante al crear la BBDD como el nombre de la MIB, el número de Objetos, Tablas y Objetos de Tabla que contiene, sus características... También contendrá el tiempo que tarda el programa en leer la MIB y el tiempo que tarda en crearse la BBDD.
- En la zona 3 (*Archivo de información*) se debe introducir la ubicación donde se desee guardar dicho informe de salida. Una vez se haya introducido toda esta información se ha de pulsar el botón 'START' para que se ejecute el programa comentado anteriormente para crear la BBDD a partir del fichero de texto introducido que contiene la definición formal de la MIB.
- En la zona 4 (*Información de salida*) se tiene en el interfaz una ventana donde irá apareciendo la información sobre el estado del programa (qué botón hemos pulsado, si se ha actualizado correctamente la información relativa a la BBDD o si se ha completado de ejecutar el programa que crea la BBDD entre otras cosas).

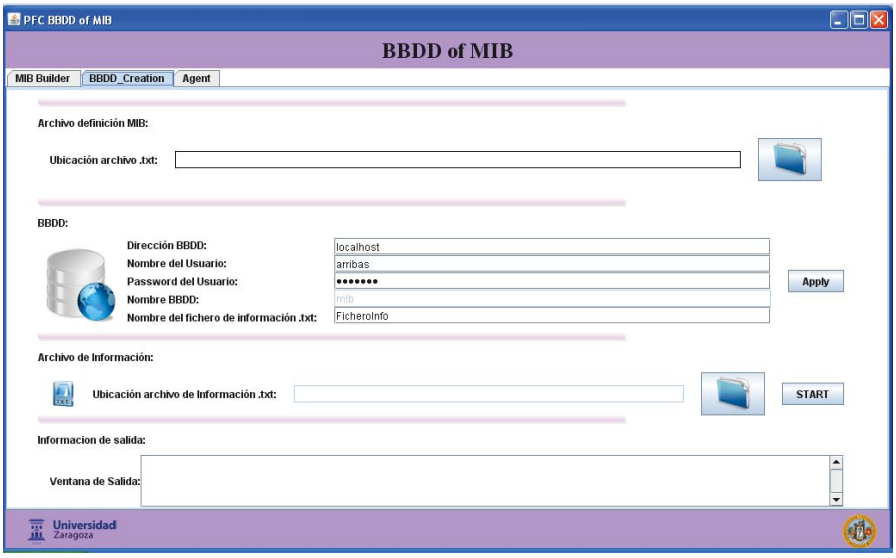

Figura 3.10: Interfaz *BBDD Creation*.

3.4 Agente SNMP

Para permitir la comunicación de un gestor con el dispositivo que contenga la MIB hay que implementar un agente SNMP. El gestor se comunicará con el agente para extraer la información almacenada en su MIB (el agente también puede modificar los valores de la MIB según se indica en el anexo D). Con el fin de facilitar al usuario final controlar el acceso al agente implementado, las cuentas de usuario necesarias para conectarse al agente pueden gestionarse a través del interfaz diseñado.

3.4.1 Implementación

La implementación del agente se ha hecho partiendo de la base de la API SNMP4j, modificando convenientemente los códigos que ya contiene, dotando al sistema de la capacidad para interactuar con cualquier gestor a través de las tres versiones que se definen en la arquitectura SNMP.

El agente desarrollado se comunica con la MIB implementada en la BBDD, y para acceder a ella, en función de la petición del gestor, se ejecuta un algoritmo u otro de lectura o de escritura.

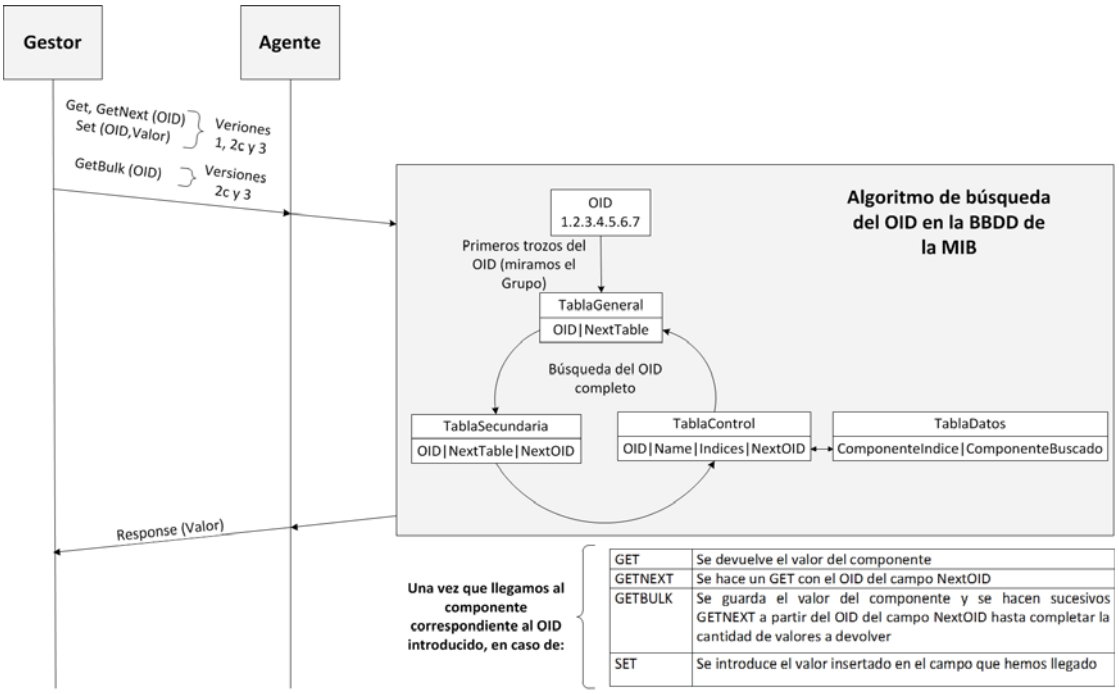

Figura 3.11: Proceso de envío y recepción de un mensaje entre gestor y agente.

En la Figura 3.11 se muestra el proceso completo desde que el gestor manda un mensaje (Get, GetNext, GetBulk o Set) en cualquier versión (1, 2c ó 3), hasta que recibe una respuesta por parte del agente, y cómo el agente busca en la BBDD el valor

correspondiente al OID. Lo primero que hace el agente es mirar en la *TablaGeneral* a qué Grupo pertenecen los primeros trozos del OID introducido. Una vez que sabe a qué Grupo pertenece, se dirige a su *TablaSecundaria* (que conoce porque está definida en la *TablaGeneral*). En la *TablaSecundaria* que ha llegado, se realiza la misma operación, es decir, compara el OID introducido (con un trozo más que antes) con los OID de los Objetos que contiene. Se continúa realizando esta operación hasta llegar al Objeto u Objeto de Tabla al que pertenece el OID completo y según sea el tipo de mensaje introducido se realiza una u otra operación. Los diagramas de flujo completos de cada tipo de mensaje posible están detallados en el anexo E. Estos diagramas de flujo han servido para desarrollar sus correspondientes algoritmos.

El interfaz de este bloque se muestra en la Figura 3.12 y puede dividirse en tres zonas:

- En la zona 1 (*Añadir usuarios del agente*) se permite definir usuarios de cualquier versión que puedan interactuar con el agente, proporcionándoles los permisos deseados.
- En la zona 2 (*Usuarios existentes*) se muestran los usuarios definidos en el agente en la ventana de salida, y también se puede eliminar un usuario seleccionándolo en el menú desplegable y pulsando el botón de eliminar usuario.
- En la zona 3 (*Configuración del agente* e *Historial de comandos*) se puede poner el puerto y la dirección IP en la que el agente va a estar escuchando. Están los botones de puesta en marcha ('Start') y detención ('Stop') del agente así como una ventana con el historial de comandos que informa de los acontecimientos sucedidos en este interfaz (usuarios añadidos o eliminados, puesta en marcha o detención del agente...).

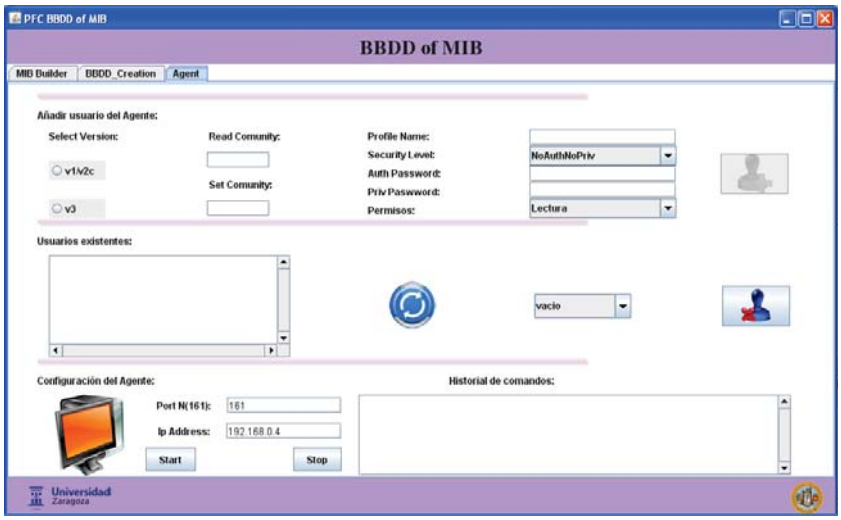

Figura 3.12: Interfaz *Agente SNMP*.

Capítulo 4

Pruebas y Resultados

Para comprobar el correcto funcionamiento del sistema se realizaron las pruebas que se recogen en este capítulo.

4.1 Diseño de una MIB

Para comprobar el correcto funcionamiento del bloque MIB Builder se ha tratado de implementar una MIB que consta de los Grupos *system* y *at* de la *mib-2*, con el fin de comparar la definición formal de la MIB introducida con los trozos correspondientes a estos Grupos de la *mib-2*. Se han elegido estos Grupos porque contienen diferentes elementos como Objetos y Tablas.

Tal y como se define en la guía de usuario del MIB Builder del anexo B, comenzamos definiendo el árbol de la MIB (que llamaremos *subMIB-2*) a partir del Grupo *mgmt*, según se ve en la Figura 4.1.

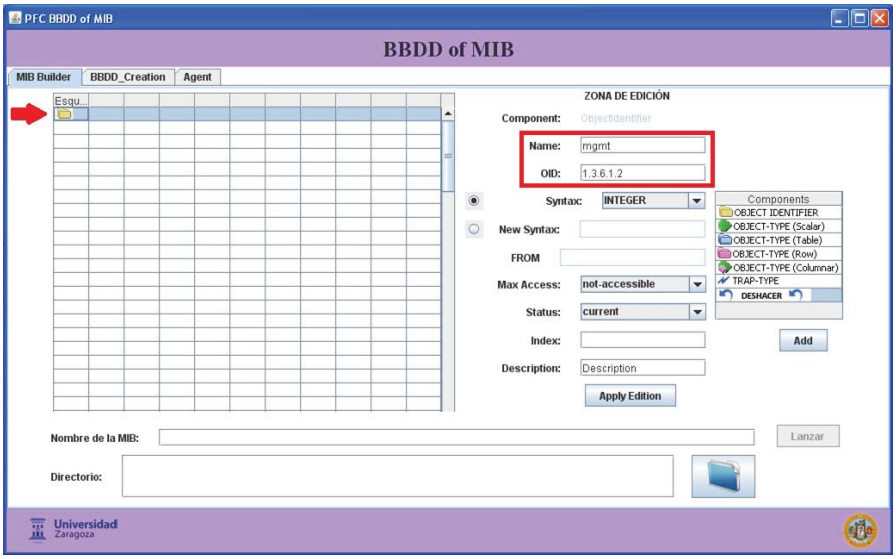

Figura 4.1: Definición del nodo inicial *mgmt*.

Una vez introducidos en el árbol de diseño todos los componentes necesarios, la estructura diseñada queda de la forma que se puede ver en la Figura 4.2, donde también se puede observar la definición del último Objeto de la Tabla *atNetAddress*, el nombre introducido para esta MIB (*subMIB-2*) y la carpeta donde se quiere guardar el fichero de texto con la definición formal de la MIB (en el escritorio):

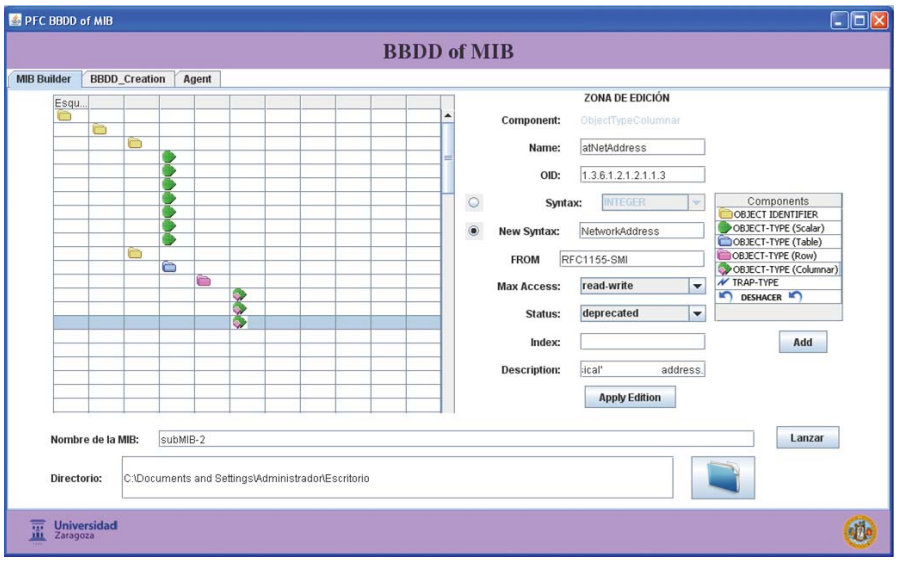

Figura 4.2: Definición del Grupo *atNetAddress*.

Ahora que ya está creado el árbol de la MIB y se han introducido los parámetros necesarios para la creación del fichero de texto de la definición formal, se ha de pulsar el botón 'LANZAR' y si todo ha funcionado correctamente nos saldrá el aviso que podemos ver en la Figura 4.3, en caso contrario, si el fichero no se hubiera generado, saldría un mensaje de error.

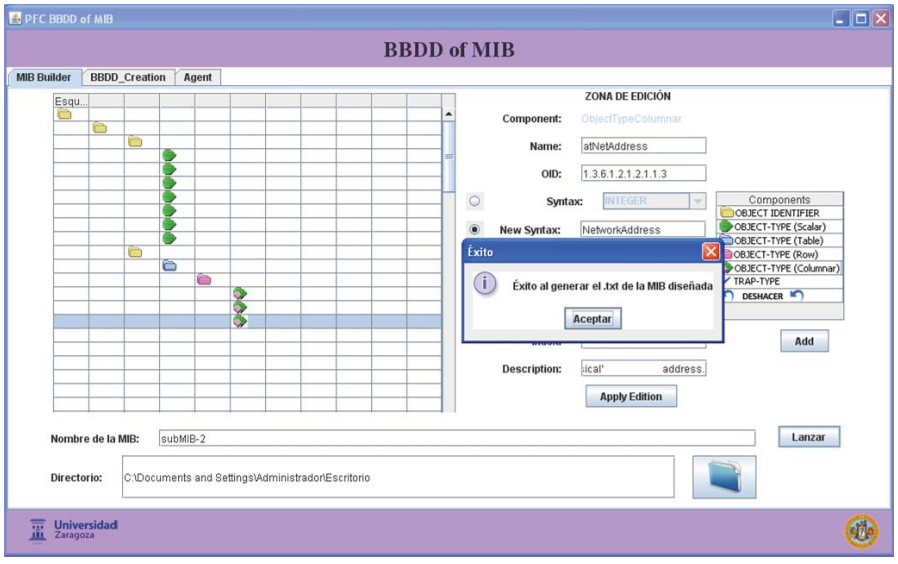

Figura 4.3: MIB definida con éxito.

Una vez creado el fichero de texto con la definición formal de la MIB, se ha comprobado su correcta definición sintáctica agregando dicho fichero a un gestor comercial como es MIB-Browser a través de su editor de MIBs (MIB Compiler [25]). Este proceso se muestra en la Figura 4.4:

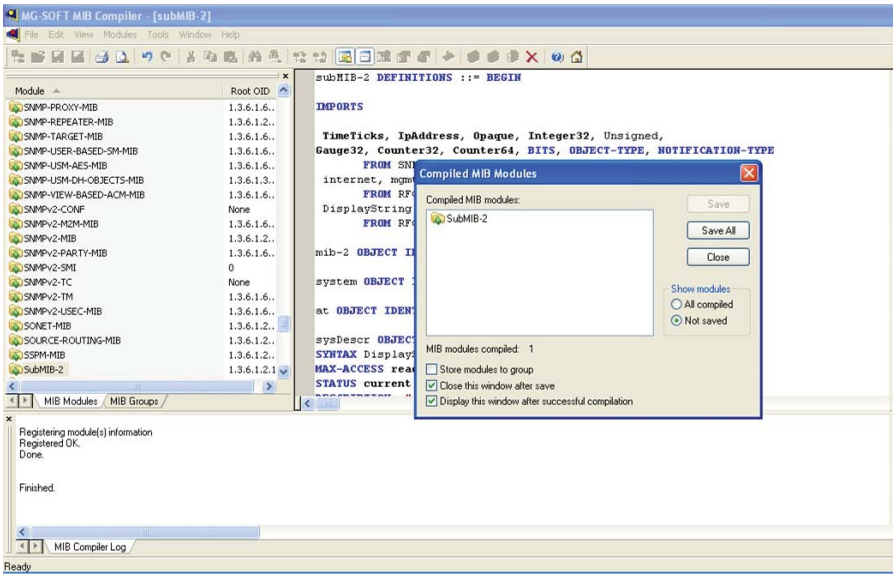

Figura 4.4: Introducción de la MIB creada en un gestor.

Y por último se muestra en la Figura 4.5 la MIB que hemos creado en el Bloque MIB Builder, en el gestor MIB-Browser:

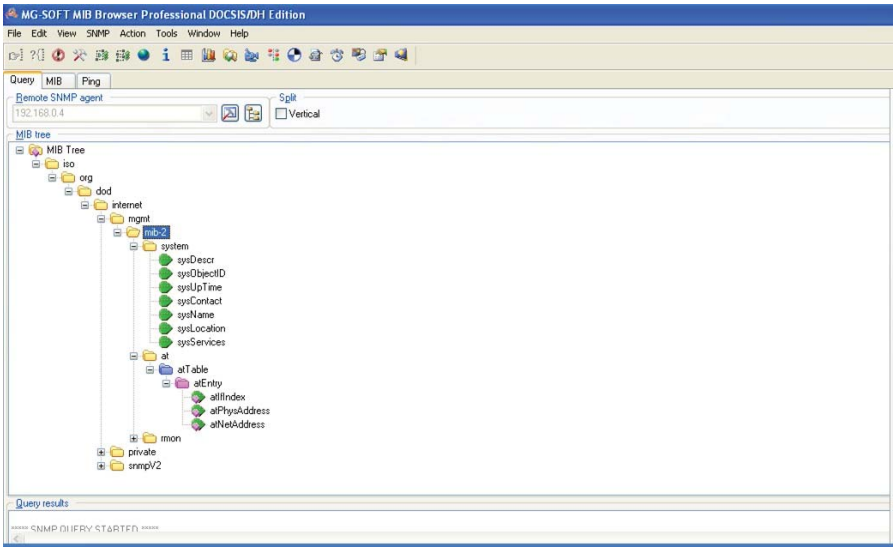

Figura 4.5: Visualización de la MIB creada en el gestor MIB Browser.

4.2 Implementación de la BBDD de una MIB

Se emplea el bloque BBDD Creation diseñado para crear la BBDD de la MIB generada con el módulo anterior. Para activar el algoritmo de implementación se han de rellenar previamente todos los campos presentes en el interfaz. Lo primero es seleccionar el archivo de texto generado en el bloque anterior (Figura 4.6). A continuación se introduce la información de configuración de la BBDD (Figura 4.7), posteriormente se selecciona la carpeta donde se guardará el archivo de salida con la información relevante obtenida del proceso de generar la BBDD y por último se ejecuta el algoritmo desarrollado para crear la BBDD pulsando el botón 'START'. En la ventana de salida del interfaz se informa de todo este proceso y de su finalización (Figura 4.8).

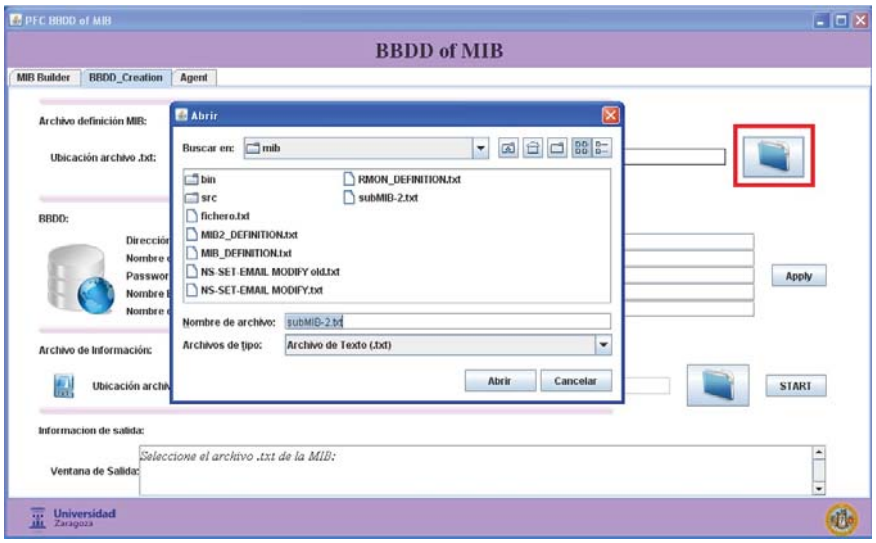

Figura 4.6: Introducción del fichero de la MIB en el bloque *BBDD Creation*.

Figura 4.7: Información de acceso a la cuenta de la BBDD MySQL.

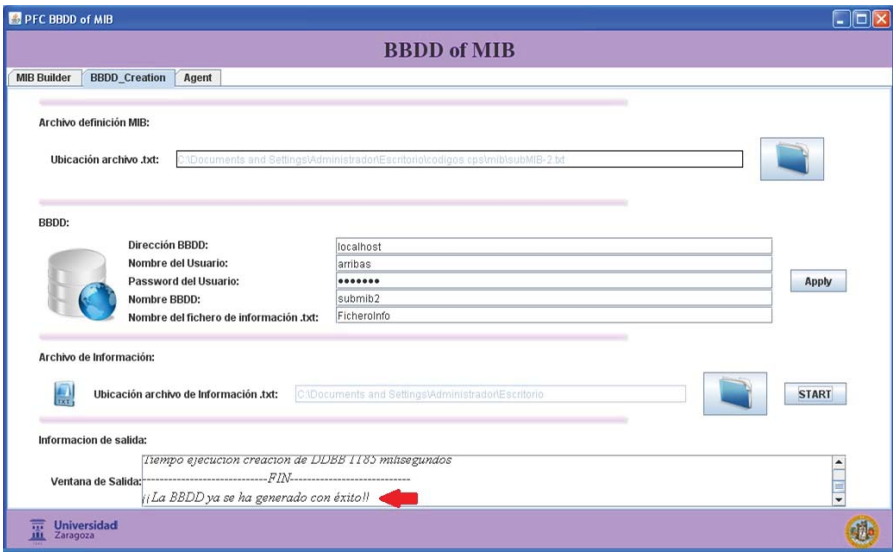

Figura 4.8: Mensaje sobre el estado de la creación de la BBDD.

4.2.1 Pruebas del bloque BBDD Creation

Con el fin de comprobar el correcto funcionamiento del bloque BBDD Creation, se ha creado un banco de pruebas para determinar con exactitud su comportamiento con los distintos tipos de MIBs existentes.

Las pruebas se han realizado en un ordenador portátil con un procesator intel core i3 en el entorno de una máquina virtual con sistema operativo Windows XP cuya memoria base era de 512 MB. Dichas pruebas han consistido en comprobar que el algoritmo de conversión de la definición formal de una MIB a una estructura de tablas para componer la BBDD funciona con todo tipo de MIBs, por lo que el banco de pruebas fue formado por las siquientes MIBs:

- *MIB-2*: MIB pública compuesta por Objetos y Tablas. Esta MIB se compone por los Grupos: *system, interfaces, at, ip, icmp, tcp, udp, egp, transmission, snmp* y *host*.

- *RMON*: MIB pública compuesta por Tablas con permisos especiales. Esta MIB se compone por los Grupos: *statics, history, alarm, hosts, hostTopN, matrix, filter, capture, event* y *rmonConformance*.

- *NETSCREEN-SET-EMAIL-MIB* [26]: MIB privada de un dispositivo de red (switch).

- *MEDICALDEVICESMANAGER-MIB* [10]: MIB privada de un dispositivo médico.

- *EjemploPrueba*: MIB privada diseñada con nuestro MIB Builder.

Además de los motivos ya expuestos, se ha elegido la 'MIB-2' para realizar las pruebas porque para el diseño de la estructura de tablas que componen la BBDD se emplearon los Grupos *system* e *interfaces* pertenecientes a esta MIB. De igual manera que de la MIB 'RMON' se utilizó el Grupo *alarm*.

Se han creado cinco BBDD en la misma cuenta de usuario MySQL (*mib*, *rmon*, *netscreen*, *medical* y *prueba*) para almacenar las BBDD implementadas al realizar las pruebas.

Una vez ejecutado el programa con las cinco MIBs que componen la prueba, obtenemos los archivos de información creados a la vez que las BBDD, nos muestran la información mostrada en la Tabla 4.1 (donde NObj es el número de Objetos, NTab el número de Tablas, NObjTab el número de Objetos de Tablas, Tobj el tiempo en reconocer los Objetos, Tcrear el tiempo en crear la BBDD y TMem el tamaño de memoria), además de información detallada de cada uno de los Objetos, de las Tablas y de los Objetos de Tabla (nombre, tipo de dato, OID...).

MIB	**NObj**	**NTab**	**NObjTab**	**Tobj (ms)**	**Tcrear (ms)**	**TMem (KB)**
MIB-2	113	8	69	251	11106	71,3
RMON	18	18	167	310	15232	108,6
NETSCREEN	5	0	0	250	451	7,8
MEDICAL	23	8	59	461	7481	48,5
EjemploPrueba	7	2	6	10	1342	16,2

Tabla 4.1: Resultados más relevantes obtenidos en las pruebas.

Puede observarse que el tiempo que tarda en crearse la BBDD (Tcrear) no es muy elevado y que el tamaño de memoria que ocupa al ser creada (TMem) es pequeño. También se puede observar que estos parámetros dependen de manera directamente proporcional al número de Objetos, Tablas y Objetos de Tabla que contengan. Hay que tener en cuenta que el tamaño de memoria de la BBDD será mayor cuando se introduzcan datos en las Tablas que contengan (ya que inicialmente están vacías).

4.3 Creación del agente

Para comprobar el correcto funcionamiento de la BBDD implementada se ha desarrollado un agente que recibirá las peticiones de información sobre la MIB de un gestor e interactuará con este. En el anexo F se muestra la guía de usuario de este interfaz.

En la Figura 4.9 se puede ver como se han introducido los usuarios *public* y *private* de la versión 1/2c, que tendrán permisos de lectura y escritura respectivamente, y se ha introducido el usuario *arribas* de la versión 3 para que puedan conectarse a nuestro agente.

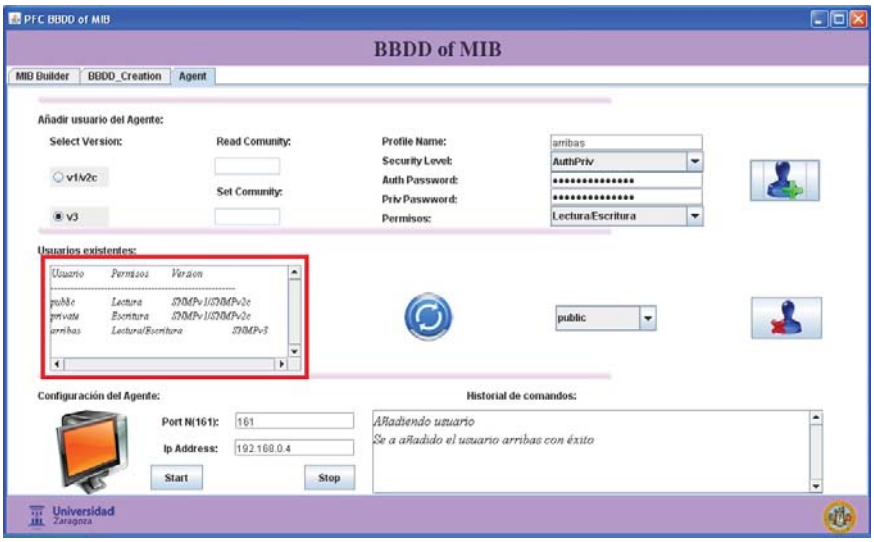

Figura 4.9: Usuarios definidos en el agente.

Una vez se han configurado las cuentas de usuario, se lanza el agente para que escuche peticiones de los gestores. Después de introducir la dirección IP del agente (la del dispositivo) y el puerto donde tiene que escuchar (normalmente el 161) se pulsa el botón 'START', y si se pone en marcha correctamente, sale un mensaje de aviso y se pone la pantalla verde del monitor del interfaz (que seguirá de ese color mientras el agente esté activo) tal y como se ve en la Figura 4.10.

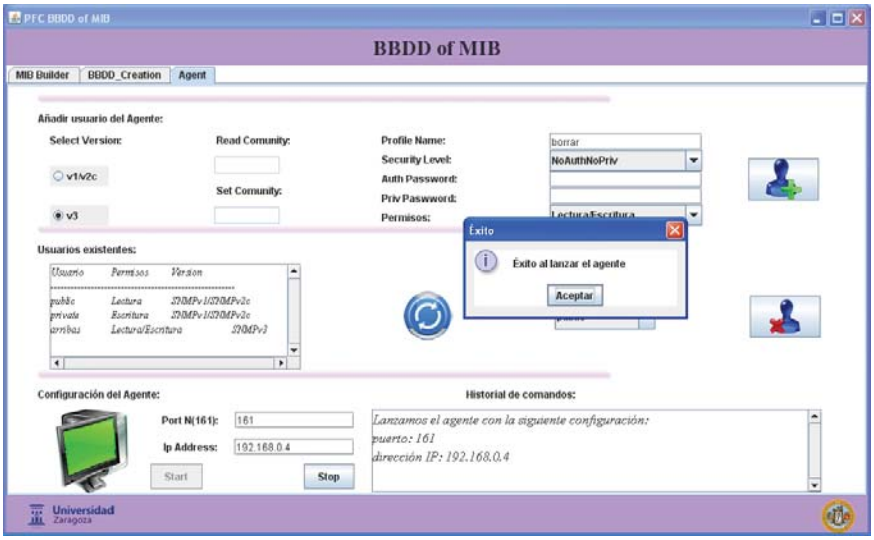

Figura 4.10: Agente funcionando correctamente.

Una vez que se tiene el agente a la escucha de los mensajes provenientes de los gestores, se dispone a verificar su correcto funcionamiento. Para ello se emplea un programa gestor (MIB-Browser) desde el cual se realizará la comunicación con el agente. Se ha de comprobar que funciona con todas las versiones del protocolo SNMP (para ello se crearon usuarios de todas esas versiones).

4.3.1 Conexión versión 1 y 2c

Puesto que la seguridad que ofrecen estas versiones es la misma, a modo de conexión es indiferente comprobar el funcionamiento de cualquiera de ellas, ya que la diferencia existente entre ellas es que con la versión 2c se pueden realizar peticiones de varios datos a la vez (GetBulk). En la Figura 4.11 se ve como se aplica la configuración de usuarios para la versión 1 y en la Figura 4.12 se ve como se ha conectado correctamente.

Figura 4.11: Configuración de usuario de la versión 1 en el gestor.

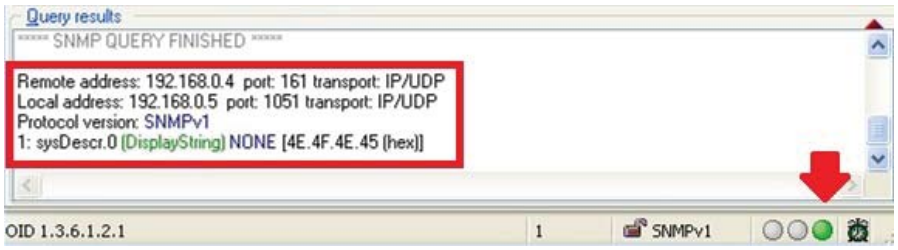

Figura 4.12: Conexión establecida entre el gestor y el agente.

Una vez que se ha conectado el gestor con nuestro agente, se comprueba que posee todas las funcionalidades correctamente implementadas. En la Figura 4.13 se muestra como se ha introducido correctamente un dato en la MIB.

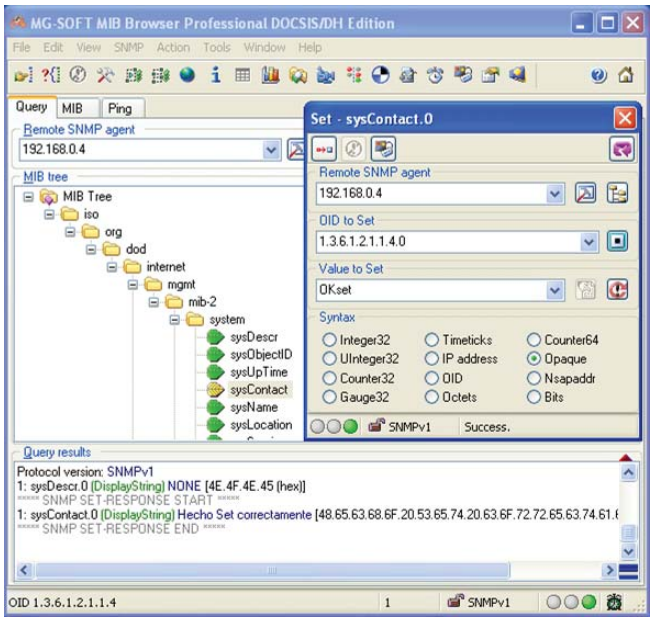

Figura 4.13: Hecho Set correctamente.

4.3.2 Conexión versión 3

Siguiendo con las pruebas del agente se pasa a verificar el correcto funcionamiento de la versión 3. Se ha de poner la información del usuario introducido en el agente para esta versión en el gestor (Figura 4.14), y en la Figura 4.15 se muestra como la conexión se ha realizado satisfactoriamente.

Figura 4.14: Configuración de usuario de la versión 3 en el gestor.

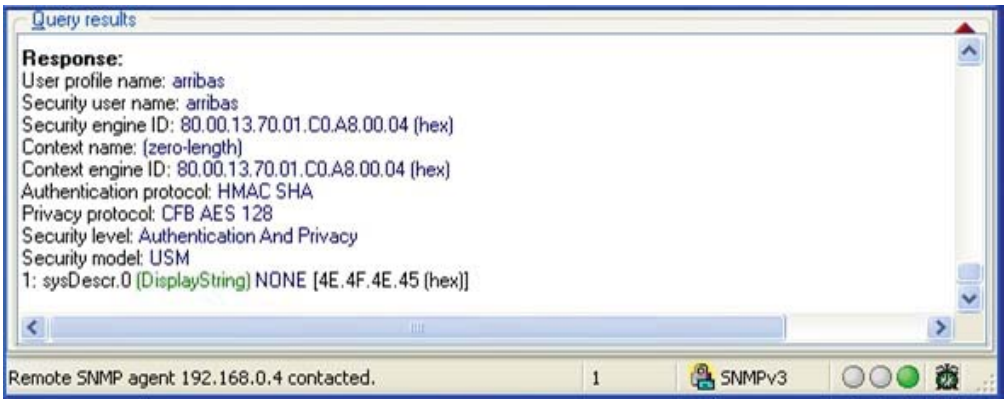

Figura 4.15: Conexión establecida entre el gestor y el agente con usuario de la versión 3.

Para seguir probando las funcionalidades del agente, se procede a leer todas las componentes del Grupo *system*, en el cual se había cambiado el valor de la componente *sysContact* por el valor 'OKset', dejando el resto con su valor inicial ('NONE'). En la Figura 4.16 se ve como todo ha salido según lo previsto.

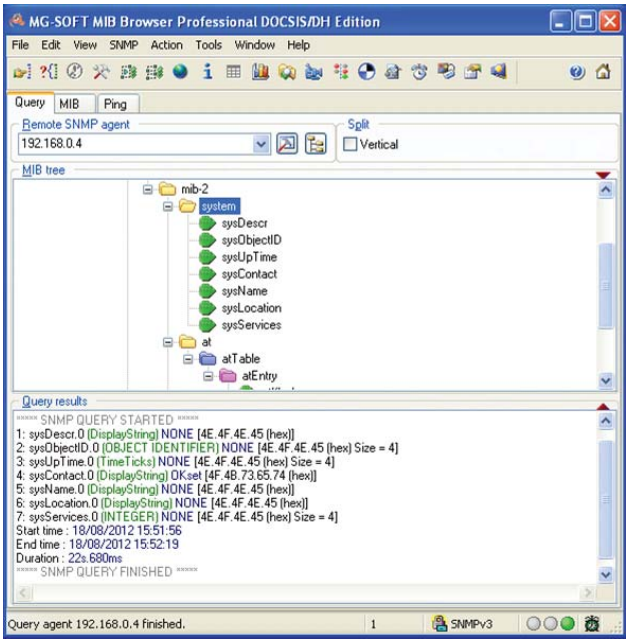

Figura 4.16: Resultado de hacer un GetBulk.

Para finalizar, una vez que se ha comprobado el correcto funcionamiento del agente se procede a detener su ejecución pulsando el botón 'STOP'.

Capítulo 5

Conclusiones y líneas futuras

5.1 Conlusiones

En este proyecto se ha desarrollado un conjunto de bloques capaces de desarrollar un agente SNMP a través de la idea inicial de una MIB. Podemos obtener la definición formal de la MIB con la ayuda del primero de los tres bloques implementados (MIB Builder), el cual nos permite obtener dicha definición formal con la ayuda de un sencillo e intuitivo interfaz gráfico. Una vez que ya se tiene la definición formal de la MIB (obtenida con nuestro MIB Builder o de cualquier otra manera), con la ayuda del segundo bloque implementado (BBDD Creation) se crea la BBDD que dará soporte al agente SNMP para almacenar la información relevante perteneciente a la MIB. Por último, el tercer bloque (Agente SNMP) será el encargado de poner en marcha el agente SNMP implementado que dará soporte a todas las versiones existentes del protocolo SNMP (1, 2c y 3) para que se pueda conectar cualquier gestor empleando cualquiera de estas versiones del protocolo.

El bloque MIB Builder constituye un sistema que ayuda a crear la definición formal de la MIB sin la ayuda del empleo de costosos programas software y sin tener que hacerlo de manera teórica, es decir, escribiendo a mano la definición formal de la MIB empleando directamente el lenguaje SMI. Esto es de gran ayuda porque además de no ser sencillo escribir directamente empleando el lenguaje SMI, es fácil cometer errores sintácticos y cuesta más hacerse una idea de como es la MIB creada que si se hace empleando un interfaz gráfico y se puede ver en todo momento la estructura de la MIB.

La utilización del algoritmo de conversión propuesto ahorrará mucho tiempo a los desarrolladores de dispositivos que usen el protocolo SNMP ya que no se tendrán que preocupar del diseño de la BBDD. Este es un proceso tedioso que conlleva gran dedicación de recursos y tiempo. Además, al no existir una estructura de BBDD estándar para el diseño de MIBs de la arquitectura SNMP, el desarrollo de las mismas y por tanto la estructura de la BBDD, se realiza de manera individual y adaptada para cada caso concreto.

Para el almacenamiento de la información, la herramienta desarrollada proporciona una estructura eficiente en la BBDD en la que se almacenan los objetos de la MIB así como un medio para la comunicación de forma sencilla con dicha MIB. La implementación física de la MIB se ha realizado mediante una BBDD MySQL y puesto que se ha de proporcionar soporte para cualquier tipo de MIB, la estructura de las tablas de la BBDD física de la MIB es genérica.

El algoritmo desarrollado permite la lectura de una MIB genérica. Por lo tanto esta herramienta podría utilizarse como base para desarrollar agentes con múltiples funcionalidades como la gestión de elementos de red (como switchs o firewalls) o de datos de usuario almacenados en un PC.

Para permitir al usuario tener todo el control sobre la BBDD y sus tablas, se deja a elección del usuario el nombre que desea darle a la BBDD para de esta manera poder tener almacenada en la misma BBDD varias MIBs, cosa que puede ser muy útil dependiendo de las necesidades que se tengan.

Cabe además destacar que el programa de conversión realizado puede servir de modelo para una futura estandarización de la estructura de la BBDD de una MIB.

El bloque destinado al desarrollo del agente se ha implementado en un interfaz (Agente SNMP) para facilitar al usuario su empleo. Permite la introducción de usuarios con sus respectivos permisos para las diferentes versiones del protocolo. También permite borrar usuarios, es decir, que se permite la gestión total de las cuentas de usuario. Se permite al usuario introducir la información de configuración del agente como es su dirección IP y el puerto por donde va a estar escuchando, y lanzar y parar el agente mediante dos sencillos botones, para que pueda interactuar a través del protocolo SNMP con cualquier gestor que tenga permisos.

Por último se han realizado pruebas mediante simulación del sistema de gestión propuesto para comprobar el alcance de la gestión del sistema y asegurar que todas las capacidades que el sistema se supone que debe cumplir para realizar una gestión completa realmente se llevan a cabo de forma satisfactoria. Mediante la consecución de estas pruebas, se presenta un resumen de los resultados obtenidos que demuestra la eficacia del sistema de gestión propuesto.

Podemos concluir que el conjunto de bloques desarrollado constituye un sistema integrado para el desarrollo de agentes SNMP partiendo de la idea inicial de una MIB. Proporciona todas las herramientas necesarias para conseguir este propósito de una manera sencilla e intuitiva, y constituye una manera de implementar agentes de una manera estándar donde la interoperabilidad con los gestores está garantizada.

5.2 Líneas de futuro

Algunas posibles líneas futuras para este proyecto serían:

- Modificar el bloque MIB Builder para permitir definir módulos MIB y tipos de datos.
- Diseñar un gestor universal que funcione con todas las versiones ya que actualmente se requiere licencia para usar las versiones 2c y 3.
- Desarrollar un interfaz para ver y modificar en tiempo real la información de la MIB como si se tratara del agente.

Bibliografía

[1] Willianm Stallings. *'SNMP, SNMPv2, SNMPv3 and RMON 1 and 2'.* Addison Wesley, 3 edition, 1999.

[2] M.Rodríguez, R.Sanz, S.Galín, C.García, R.Chinchilla, and A.Yela. 'CORBA: Una plataforma software para el futuro'. *ISA España.*

[3] Shinkyung Lee and Dooho Choi. Namje Park. 'Security Mangement System for Sensor Network'. *International Conference of Convergence and Hybrid Information Technology 2008*, 2008.

[4] XML. *http://es.wikipedia.org/wiki/Extensible_Markup_Language.*

[5] MG-SOFT Corporation. *http://www.mg-soft.si/.*

[6] NuDesing Technologies. *http://www.ndt-inc.com/.*

[7] Yoshiaki Kiriha, Shoichiro Nakai, Keiko Aroma, and Hiroshi Kuriyama. 'An Automatic Generation of Management Information Base (MIB) for OSI based Network Management System'. *Global Telecommunications Conference, 1991. GLOBECOM '91. 'Countdown to the New Millennium. Featuring a Mini-Theme on: Personal Communications Services*, 1:649–653, dec. 1991.

[8] Yoshiaki Kiriha. 'Real-Time MIB: Implementation and Evaluation'. *NEC C&C Research Laboratories*, pages 608–611, 1996.

[9] N. Lasierra, M. Lopez, J. García, and A. Alesanco. 'Desarrollo de un agente SNMP v3 para modelado de usuario en entornos LAN'. *VIII Jornadas de Ingeniería Telemática JITEL*, pages 397–404, 2009.

[10] N. Lasierra, A. Muñoz, A. Alesanco, J. Escayola, and J. García. 'Una solución SNMP para la gestión técnica de dispositivos médicos ISO/IEE 11073'. *XXIX Congreso Anual de la Sociedad Española de la Ingeniería Biomédica*, pages 255–258, 2011.

[11] José García and Álvaro Alesanco. 'Web-Based System for Managing a Telematics Laboratory Network'. *IEEE TRANSACTIONS ON EDUCATION*, 47(2):284–294, May 2004.

[12] MG-SOFT Agent Builder. *http://www.mg-soft.si/builder.html*.

[13] The Eclipse Foundation. *http://www.eclipse.org/*.

[14] MG-SOFT MIB Browser. *http://www.mg-soft.si/mgMibBrowserPE.html*.

[15] MySQL. *http://www.mysql.com/*.

[16] API-SNMP4J. *http://www.snmp4j.org/*.

[17] MIB-2. *http://www.ietf.org/rfc/rfc1213.txt*.

[18] William Stallings. 'SNMPv3: A Security Enhancement for SNMP'. *IEEE Communicatios Surveys*, 1(1):2–17, 1998.

[19] Storage Networking Industry Association. *http://www.snia.org/*.

[20] Storage Management Forum. *http://www.snia.org/forums*.

[21] Technical Work Groups. *https://www.snia.org/tech_activities/work/twgs*.

[22] Conformance Testing Program. *http://www.snia.org/ctp/*.

[23] RFC 2578: SMIv2. *http://tools.ietf.org/html/rfc2578*.

[24] RMON. *http://tools.ietf.org/html/rfc2819*.

[25] MG-SOFT Mib Compiler. *http://www.mg-soft.si/mgmibc.html*.

[26] Juniper Networks. *http://www.juniper.net/au/en/products-services/security/netscreen/*.