Índice general

A Manual Agente SNMPv3 **1**

A.1 Introducción 1

A.2 Configuración y puesta en marcha del agente SNMPv3 1

A.2.1 Ventana de configuración 1

A.3 Ejemplo práctico 4

B Manual Interfaz Proxy **7**

B.1 Introducción 7

B.2 Descripción del interfaz 7

C Descripción de objetos de las MIB **13**

C.1 Grupo ComputeEngineControlInfo 13

C.2 Grupo MedicalDeviceInfo 15

C.2.1 MedicalDeviceControlTable 15

C.2.2 MedicalDeviceDataTable 16

C.2.3 MedicalDeviceStateTable 17

C.2.4 SpecificErrorsTable 18

C.3 Grupo AlarmTable 19

C.4 Grupo EventTables 20

C.4.1 ConfigEventTable 20

C.4.2 LogTable 21
C.5 Grupo ManagerTable 21

D Estructura interna de la MIB MedicasDevicesManager 23
D.1 Estructura interna de tablas para una MIB genérica 23
D.1.1 Estructura genérica de una tabla inicial 23
D.1.2 Estructura genérica de una tabla de control 26
D.1.3 Estructura genérica de una tabla de instancias 28
D.1.3.1 Tabla de instancias de control 28
D.1.3.2 Tabla de instancias de datos 29
D.2 MySQL 29
D.3 Organización interna de la MIB Medical Devices Manager 30

E Definición formal de la MIB Medical Devices Manager 33

F Diagrama de Gantt 69

Índice de figuras

A.1 Interfaz gráfico del agente SNMPv3 2
A.2 Ventana de error al escribir mal el puerto 4
A.3 Interfaz gráfico funcionando 5
A.4 Ventana de configuración SNMPv3 de MIB Browser 6
A.5 Agente conectado con éxito al gestor SNMPv3 6

B.1 Aspecto del interfaz proxy SNMP-X.73 8
B.2 Pestañas del interfaz proxy SNMP-X.73 10
B.3 El dispositivo no esta en estado AVAILABLE todavía 11
B.4 El dispositivo no esta en estado OPERATING 12

F.1 Diagrama de Gantt del Proyecto Fin de Carrera 71

Índice de tablas

D.1 Definición de tabla inicial 24
D.2 Ejemplo de tabla inicial: MIB interfaces 25
D.3 Definición de tabla control 26
D.4 Ejemplo de tabla de control: tabla ifTable 27
D.5 Definición de la estructura de una tabla de instancias de control . . 28
D.6 Ejemplo de tabla de instancias: tabla ifSpeed 28
D.7 Definición de la estructura de una tabla de instancias de datos . . . 29
D.8 Tabla de MySQL que controla la tabla CEControlTable 31
D.9 Tabla de MySQL que controla la tabla MDControlTable 31
D.10 Ejemplo de una tabla de instancias de datos para el objeto IdState 32
D.11 Ejemplo de tabla de instancias de control para el objeto SystemId . 32

Anexo A

Manual Agente SNMPv3

A.1 Introducción

En este anexo se presenta una guía introductoria al uso y configuración del agente SNMPv3. Está enfocada a los gestores del sistema, describiendo los pasos que deben dar para realizar una correcta conexión con el agente. La interfaz gráfica se ha programado utilizando Java, mediante la herramienta Swing. Para la correcta ejecución del GUI (*Graphic User Interface*) es necesario tener instalada la máquina virtual de Java versión 1.6 o superior.

A.2 Configuración y puesta en marcha del agente SNMPv3

A.2.1 Ventana de configuración

Para lanzar el agente disponemos de un archivo ejecutable creado a partir de las clases de Java que han sido utilizadas para su programación. Una vez lanzamos el agente, nos parece una ventana de configuración que nos permite seleccionar los parámetros que nos permiten establecer el nivel de seguridad de la transmisión. Como se puede observar en la figura A.1, que muestra el interfaz del usuario, disponemos de una serie de botones y cuadros de texto para realizar la

configuración. El interfaz está dividido en dos partes; en la parte superior podemos realizar la configuración del usuario, mientras que en la inferior encontramos la configuración del agente.

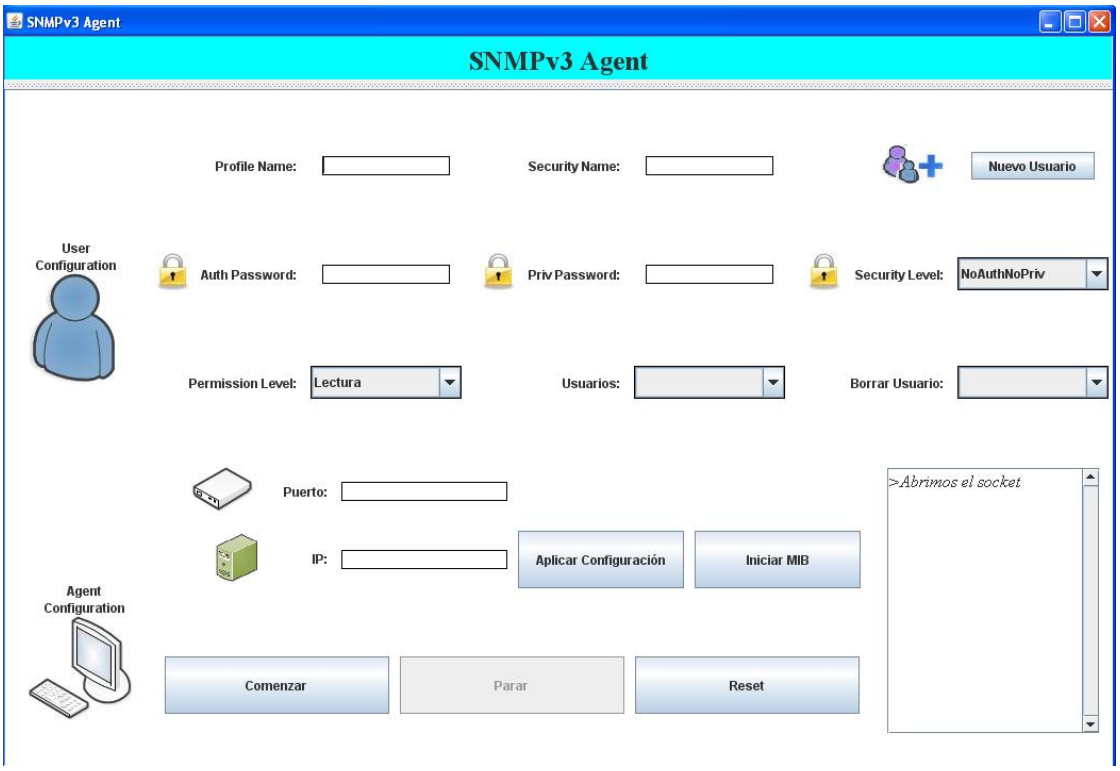

Figura A.1. Interfaz gráfico del agente SNMPv3.

Para la correcta configuración del usuario debemos rellenar los siguientes parámetros:

- Profile Name: Es el nombre que usaremos para identificar el perfil de configuración de cada usuario.
- Security Name: Consiste en el nombre de usuario del protocolo de seguridad USM de SNMPv3.
- Auth Password: Contraseña utilizada para autenticar el mensaje, utilizada en este caso con la función hash SHA.
- Priv Password: Contraseña utilizada para garantizar la privacidad del mensaje. En este caso se utiliza cifrado AES con contraseña de 128 bits.

- Security Level: Parámetro que establece el nivel de seguridad que vamos a utilizar en las comunicaciones SNMPv3. Las opciones son las siguientes: *AuthPriv* para autenticación y cifrado; *AuthNoPriv* para autenticación sólamente; *NoAuthNoPriv* para la comunicación sin mecanismos de seguridad.

- Permission Level: Privilegios del usuario dentro de la MIB. Pueden ser de lectura o de lectura/escritura.

Además de los parámetros necesarios para establecer la seguridad de SNMPv3, hemos incluido otros que utilizaremos para realizar gestión de usuarios. De este modo, cuando alguien desee establecer una configuración, tiene la opción de guardar esa configuración en una tabla de modo que en el futuro no tenga que volver a escribirla. Mediante el botón *Nuevo Usuario* guardamos la nueva configuración, siendo obligatoria la definición de un nombre de perfil no existente ya en la tabla. Si queremos cargar una configuración ya existente, en el menú desplegable *Usuarios* podemos observar todas los nombres de perfiles ya existentes. Basta con elegir el perfil deseado y sus valores se cargan en los cuadros de texto listos para ser aplicados. En caso de que queramos eliminar una configuración, al elegirla en el menú *Borrar Usuario* se eliminará de la tabla de usuarios.

Para garantizar que los datos introducidos en los cuadros de texto son del formato adecuado, se han dispuesto una serie de mensajes de error en caso de que alguno de ellos no sea adecuado. Concretamente, aparece una ventana de error si el puerto no es un número entero (como vemos en la figura A.2) o si las contraseñas de autenticación y privacidad no tienen un mínimo de 8 caracteres. Tambíen nos aparecerá una ventana cuando queramos establecer un nuevo usuario y el nombre de perfil ya exista en la base de usuarios.

Para realizar la configuración del agente, además de configurar el usuario, se debe establecer un puerto para que el agente espere la conexión del gestor. Cuando todo esté correctamente configurado, se aplicarán los parámetros seleccionados pulsando el botón *Aplicar Configuración*. Antes de lanzar el agente, debemos

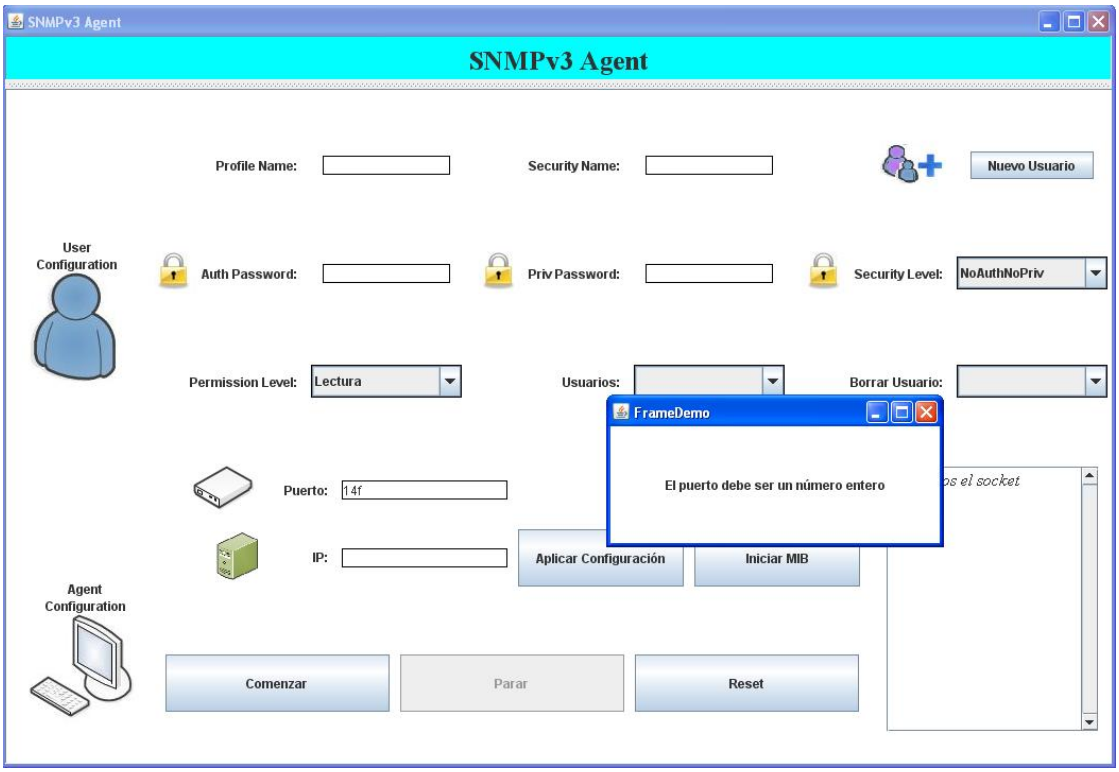

Figura A.2. Ventana de error al escribir mal el puerto.

asegurarnos de que la MIB está operativa. Pulsando *Iniciar MIB*, se crearán las tablas necesarias en caso de que no existan. Si en algún momento se quiere borrar las tablas de la MIB, pulsando el botón *Reset* se borran todas las tablas de datos y se crean de nuevo las tablas de control iniciales. Destacar que el reseteo de la MIB no borra la tabla de gestión de usuarios.

Cuando todo lo anterior esté dispuesto, se podrá inciar el agente de forma correcta mediante el botón *Comenzar*. Para parar el agente, se pulsará el botón *Parar*.

A.3 Ejemplo práctico

Consideremos que vamos a lanzar el agente y va a estar escuchando en la dirección IP 192.168.0.4 en el puerto 161 UDP, que es el puerto de SNMP por defecto. Para ello, vamos a crear un nuevo usuario con los siguientes parámetros:

- Profile Name: usuario

- Security Name: usuario
- Auth Password: authpassword
- Priv Password: privpassword
- Security Level: AuthPriv
- Permission Level: lectura

Para configurar el agente seguimos los siguientes pasos:

1. Iniciamos el agente haciendo clic en el ejecutable.
2. Introducimos los parámetros que hemos seleccionado tal como se aprecia en la figura A.3. Una vez introducidos todos le damos a *Aplicar Configuración*.
3. Iniciamos el agente pulsando en *Comenzar*.

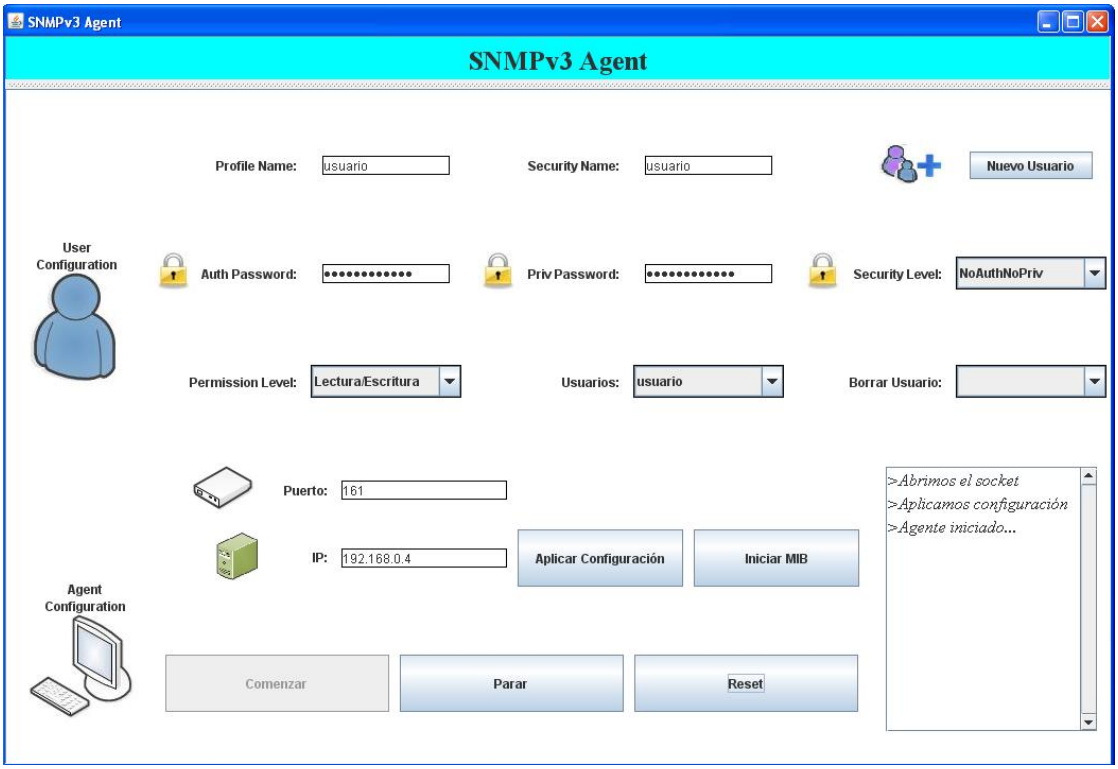

Figura A.3. Interfaz gráfico funcionando.

Una vez listo el agente, comprobamos que el agente está correctamente configurado utilizando la herramienta para gestión SNMP *MG-Soft MIB Browser*.

Como se aprecia en la figura A.4, primero creamos un usuario en el gestor con los mismos datos que el usuario del agente. Finalmente, en la figura A.5 se observa que el gestor ha conseguido conectarse con el agente SNMPv3.

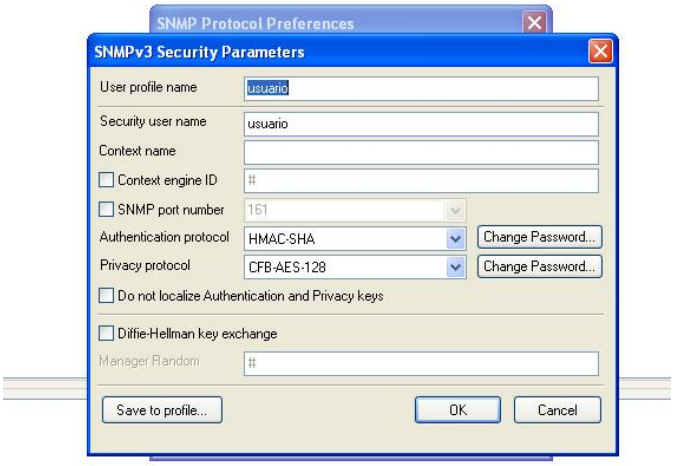

Figura A.4. Ventana de configuración SNMPv3 de MIB Browser.

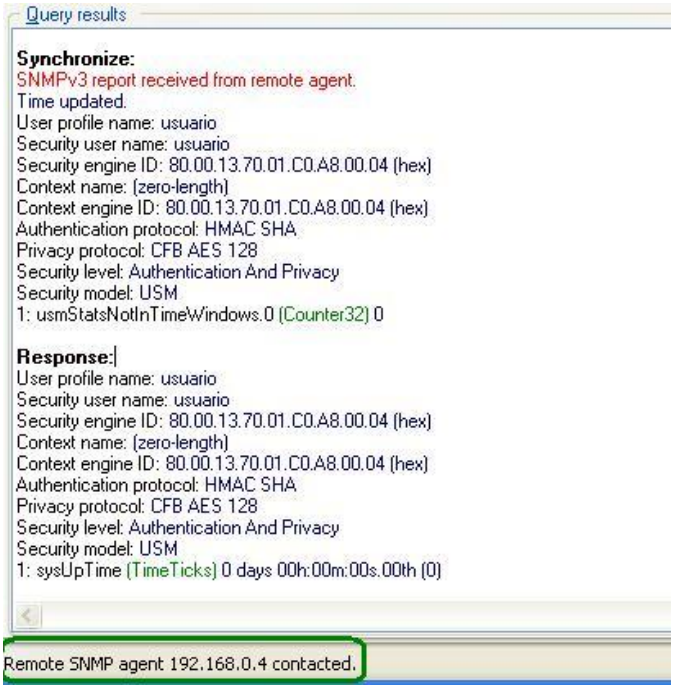

Figura A.5. Agente conectado con éxito al gestor SNMPv3.

Anexo B

Manual Interfaz Proxy

B.1 Introducción

En este anexo se presenta una guía para la configuración y funcionamiento del interfaz diseñado para simular el manager X.73 que se encarga de recoger la información de los dispositivos médicos. La interfaz gráfica se ha programado utilizando Java, mediante la herramienta Swing. Para la correcta ejecución del GUI (*Graphic User Interface*) es necesario tener instalada la máquina virtual de Java versión 1.6 o superior.

B.2 Descripción del interfaz

Este interfaz va a simular el funcionamiento del manager X.73, por lo tanto se va a encargar de crear los documentos XML pertinentes. Será el interfaz gráfico el que nos permita seleccionar el tipo de dispositivo, tipo de medida y los valores de la misma. En la figura B.1 se muestra el aspecto del interfaz. Como se puede apreciar, disponemos de 4 pestañas, una para cada dispositivo médico. Podemos ver cada una de las pestañas en la figura B.2. Cada pestaña está dividida en 3 secciones. La superior nos permite simular la conexión y desconexión de un MD; la central se utiliza para configurar una medida técnica, mientras que la inferior nos permite cambiar el estado de conexión del MD. A continuación se explicará con más detalle cómo se utiliza el interfaz.

Para empezar, recordar que este interfaz simula toda la información que el manager recibe de los MDs, así como la forma de transmisión del protocolo X.73. Es decir, para poder enviar un mensaje como si fueramos el manager tenemos que respetar el orden que se sigue en X.73. Un dispositivo no puede enviar un dato técnico en el primer mensaje, antes tiene que conectarse al CE, e ir cambiando sus estados de conexión progresivamente hasta llegar al adecuado para poder enviar el mensaje.

Los pasos que debe seguir un MD para realizar una transmisión son los siguientes:

1. Conexión del MD con el manager, lo que implica enviar un estado AVAILABLE.

2. El dispositivo pasa por diferentes estados que pueden variar dependiendo de cómo se realice la conexión, para finalmente llegar a ASSOCIATED, estado

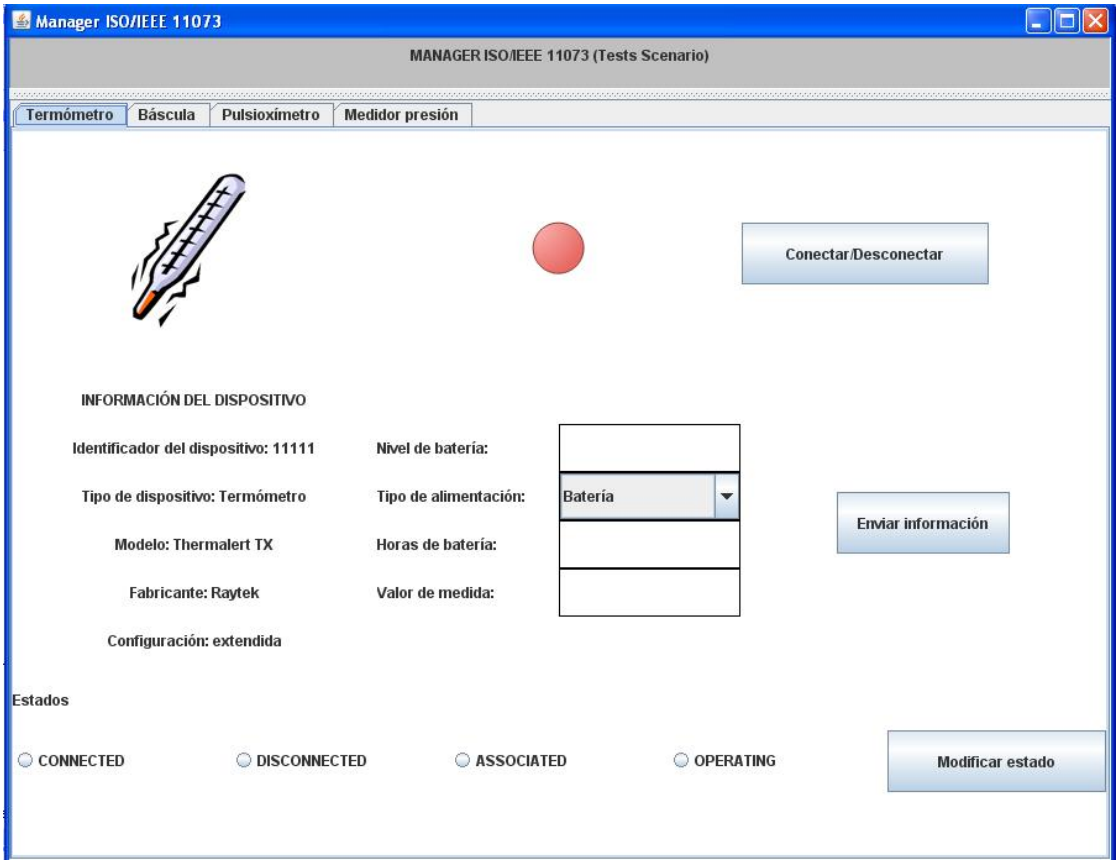

Figura B.1. Aspecto del interfaz proxy SNMP-X.73.

que implica que el dispositivo está conectado y reconocido por el manager, listo para enviar.

3. Cuando el dispositivo está en estado ASSOCIATED y quiere enviar información, pasa a estado OPERATING y envía la información requerida.

A la hora de utilizar el interfaz vamos a respetar este esquema de comunicaciones. Por lo tanto, no podremos darle un estado al dispositivo hasta que no lo hayamos conectado al manager mediante el botón *Conectar/Desconectar*. Si el botón que se encuentra a la izquierda está en color rojo, implica que el dispositivo está desconectado y al darle al botón lo conectaremos. Si está en verde implica que está conectado y al darle al botón lo desconectaremos.

Una vez lo hayamos conectado, podremos cambiar el estado del dispositivo seleccionando el que queramos de la lista existente y presionando el botón *Modificar Estado*. Si ahora queremos simular la emisión de un dato técnico, debemos asegurarnos que el último estado que hemos enviado es OPERATING. En otro caso no se nos permitirá enviar la medida. En las figuras B.3 y B.4 se observa cómo se controla la correcta simulación del manager mediante el control del orden de los estados.

En la configuración de la información técnica, los datos estáticos del dispositivo tales como fabricante, modelo o identificador están fijados a valores comerciales, ya que en la realidad esos valores tampoco cambiarán con las diferentes medidas. Los valores que sí podemos modificar son el nivel de batería, tipo de alimentación, horas de batería restantes y valor del dato médico. En el caso del pulsioxímetro disponemos también de un lista de posibles errores específicos que sólo afectan a este dispositivo. Sólo se admiten valores numéricos para las datos configurables. Una vez tengamos la entrada a punto, la enviamos presionando el botón *Enviar Información*.

El funcionamiento del interfaz aquí explicado se mostrará de forma gráfica mediante capturas de pantalla en el apartado de resultados, ya que se utilizará para el envío de información al agente.

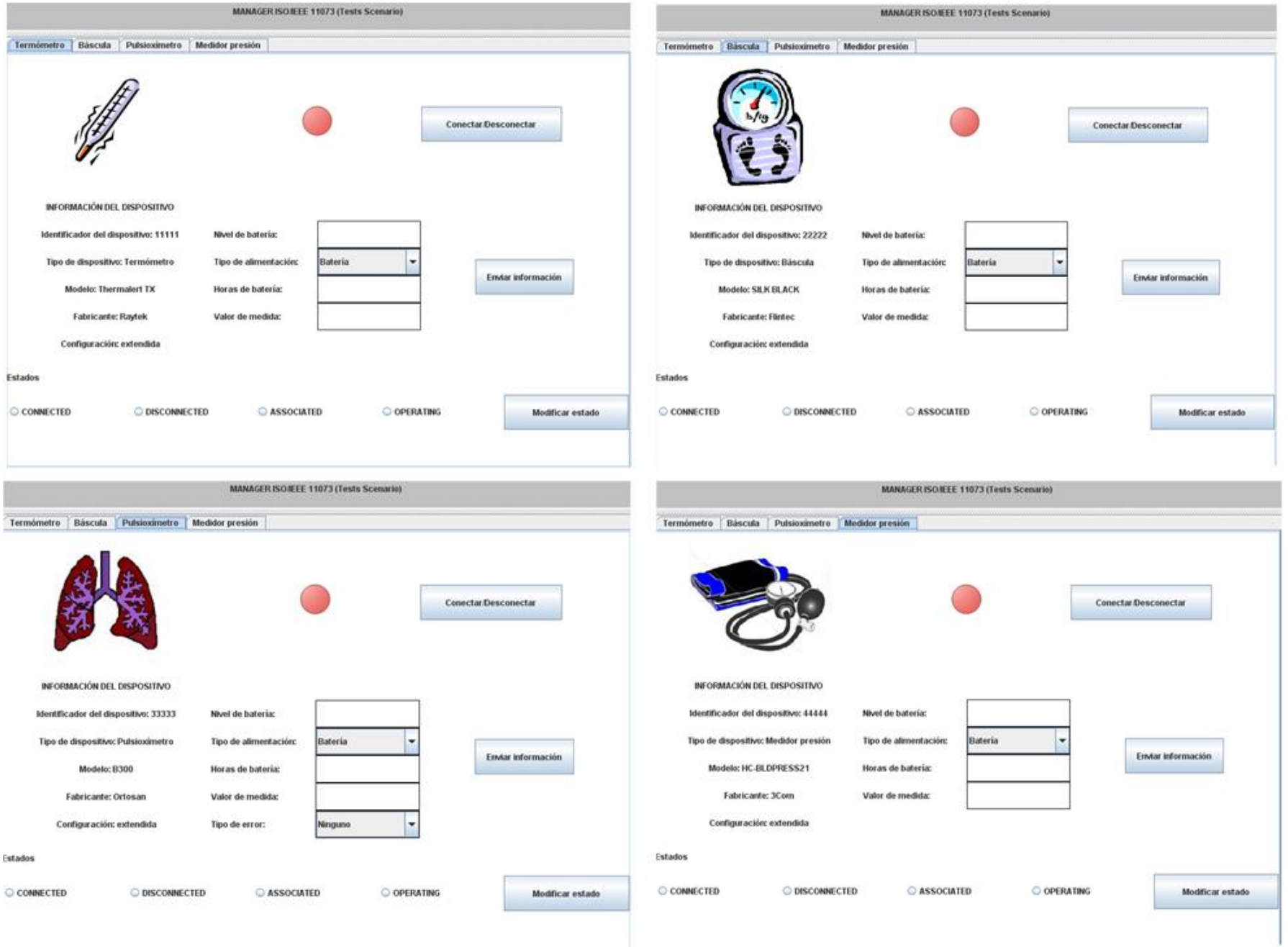

Figura B.2. Pestañas del interfaz proxy SNMP-X.73.

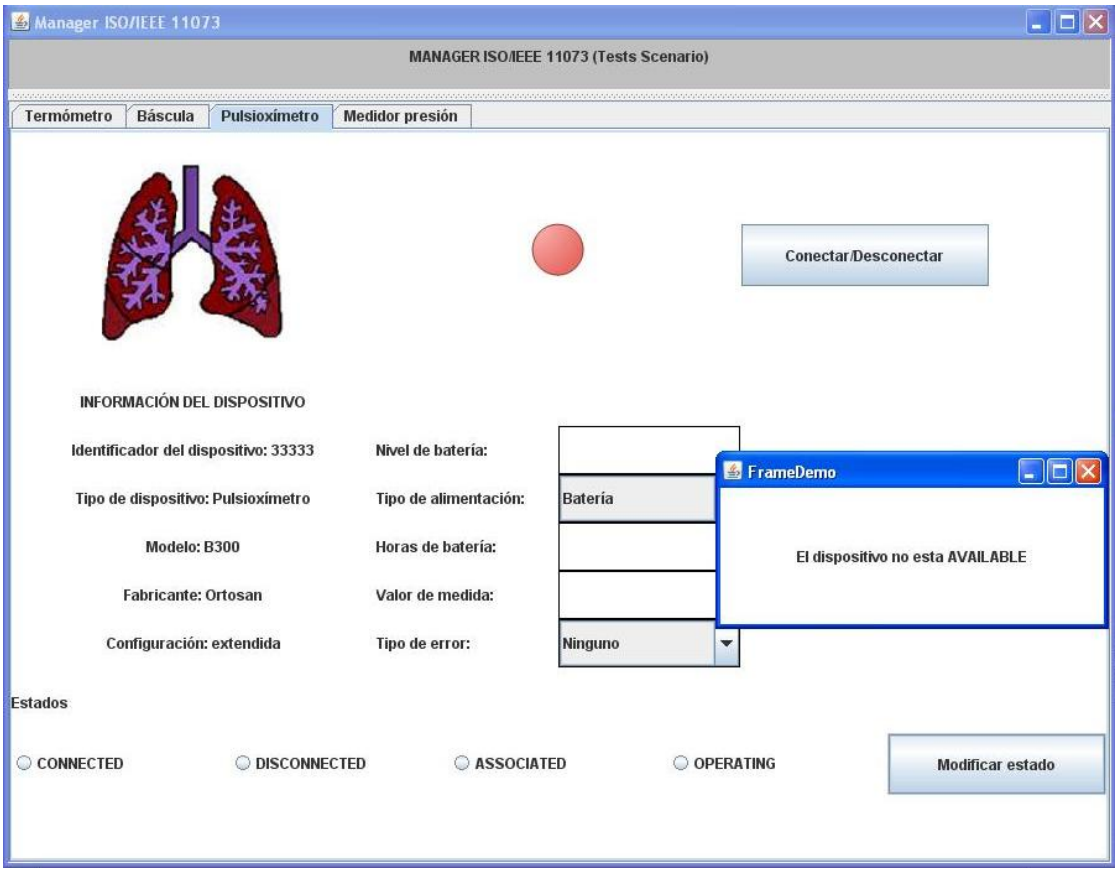

Figura B.3. El dispositivo no esta en estado AVAILABLE todavía.

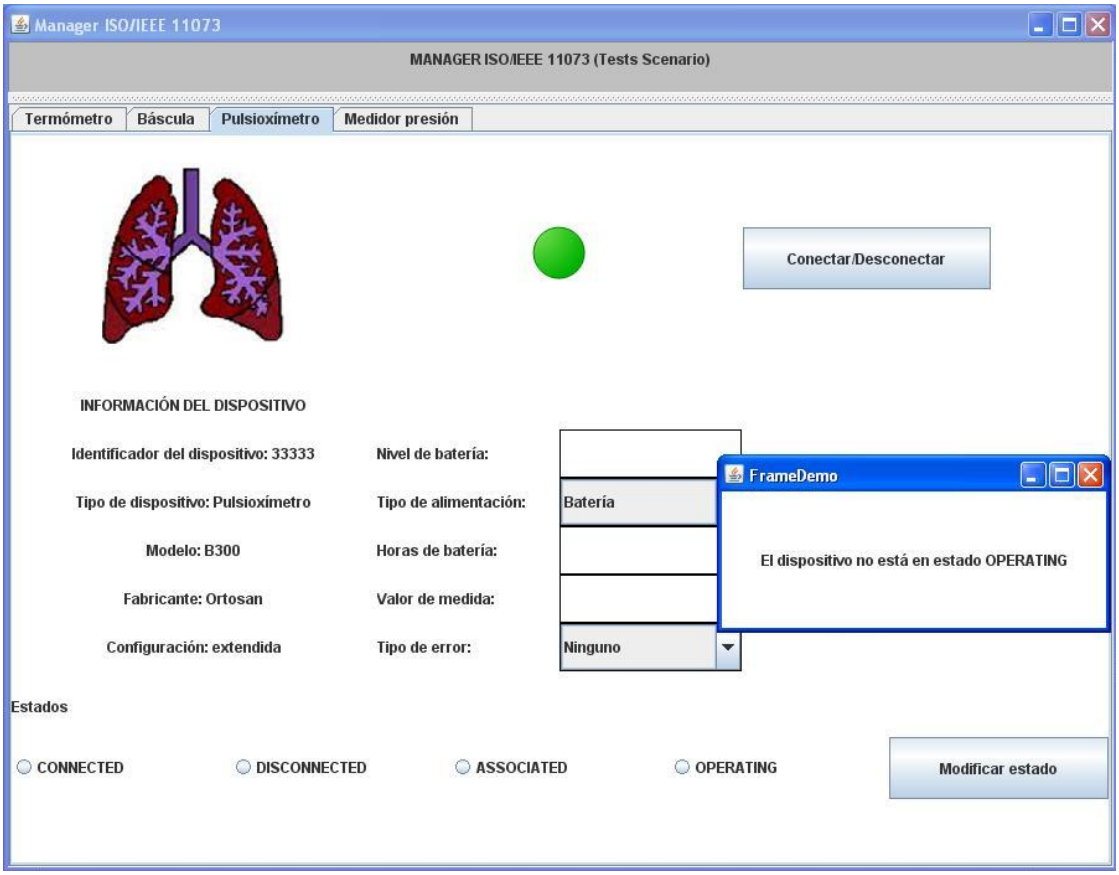

Figura B.4. El dispositivo no esta en estado OPERATING.

Anexo C

Descripción de objetos de las MIB

Para una mejor visualización de la estructura jerárquica de la MIB, se hace uso del interfaz gráfico que nos proporciona el gestor que utilizamos para comprobar el funcionamiento del agente, el programa *MIB Browser* de *MG-Soft*. Este programa proporciona, además de un interfaz gráfico que facilita la comprensión de la organización estructurada de los campos de la MIB, una herramienta para la gestión de redes que permite el envío de paquetes SNMP así como una pantalla de resultados en la que se visualizan los efectos y consecuencias de las peticiones realizadas. A pesar de no ser la única herramienta disponible para enviar paquetes SNMP ni leer el contenido de la MIB, se elige trabajar con ella dada su facilidad de uso e interfaces gráficos que muestran y facilitan el entendimiento de la organización de las MIBs. Además de este programa se utiliza el programa *MIB Run Compiler* para partir de las definiciones formales realizadas de las MIBs creadas, obtener la visualización de todas ellas en el interfaz gráfico que proporciona el *MIB Browser*.

C.1 Grupo ComputeEngineControlInfo

Los campos perteneciente a este grupo son los siguientes:

- **IdComputeEngine (1.3.6.1.4.1.28308.4.1.1):** Identificador del dispositivo concentrador CE dentro del sistema.

- **DeviceType (1.3.6.1.4.1.28308.4.1.2):** Indica el tipo de dispositivo que es el CE, por ejemplo una PDA o un portátil.

- **WorkingState (1.3.6.1.4.1.28308.4.1.3):** Indica si el funcionamiento del CE está siendo correcto. Este estado puede ser operativo o dañado.

- **CommunicationState (1.3.6.1.4.1.28308.4.1.4):** Este parámetro indica si el CE está mandando información, recibiendo información o en estado *idle*.

- **AsociatedMDs (1.3.6.1.4.1.28308.4.1.5):** Indica el número de dispositivos médicos que se han registrado en este CE.

- **BandwidthUse (1.3.6.1.4.1.28308.4.1.6):** Muestra en tanto por cierto el ancho de banda utilizado por el CE del total disponible. El control de este valor nos permite evitar problemas de congestión.

- **CpuUse (1.3.6.1.4.1.28308.4.1.7):** Muestra en tanto por cierto el uso de procesador por parte del CE. Su control permite evitar sobrecargar de procesos el CE y por lo tanto ralentizarlo.

- **HardDiskMemoryUse (1.3.6.1.4.1.28308.4.1.8):** Muestra en tanto por ciento la cantidad de disco duro utilizada del CE. El control de esta variable nos permite conocer la cantidad de espacio libre en el disco duro y evitar quedarnos sin espacio para almacenamiento.

- **VirtualMemoryUse (1.3.6.1.4.1.28308.4.1.9):** Muestra en tanto por ciento la cantidad de memoria virtual utilizada por el CE. Su control permite evitar la falta de memoria virtual.

- **PhysicalMemoryUse (1.3.6.1.4.1.28308.4.1.10):** Muestra en tanto por ciento la cantidad de memoria física, o memoria RAM, utilizada por el CE. Su control permite evitar la falta de memoria física y por tanto el ralentizamiento

del procesador, pudiendo llegar a bloquear algunos de los procesos que se estuvieran ejecutando.

- **UserContactInformation (1.3.6.1.4.1.28308.4.1.11):** Incluye información acerca del usuario que utiliza este CE para poder contactar con él en caso de ser necesario.

- **HostIPAddress (1.3.6.1.4.1.28308.4.1.12):** Muestra la dirección IP del CE dentro de la red a la que pertenece.

- **UpdateRequestStateMDs (1.3.6.1.4.1.28308.4.1.13):** Esta variable sirve para pedir actualizaciones del estado de todos los dispositivos registrados en el CE.

- **UpdateRequestTechnicalDataMDs (1.3.6.1.4.1.28308.4.1.14):** Esta variable sirve para pedir actualizaciones de la información técnica de todos los dispositivos registrados en el CE.

- **dateTimeControl (1.3.6.1.4.1.28308.4.1.15):** Indica la fecha en que fue realizada la última actualización de los recursos propios del CE.

C.2 Grupo MedicalDeviceInfo

C.2.1 MedicalDeviceControlTable

Los parámetros de esta tabla se describen a continuación:

- **IdMDControl (1.3.6.1.4.1.28308.4.2.1.1.1):** Identificador que se le asigna a cada MD dentro de nuestra MIB.

- **Manufacturer (1.3.6.1.4.1.28308.4.2.1.1.2):** Indica el nombre del fabricante del MD.

- **Model (1.3.6.1.4.1.28308.4.2.1.1.3):** Indica el modelo del MD dentro de la gama del fabricante.

- **SystemId (1.3.6.1.4.1.28308.4.2.1.1.4):** Identificador del MD. Este valor se lo proporciona el fabricante para identificar ese dispositivo en concreto.

- **MDType (1.3.6.1.4.1.28308.4.2.1.1.5):** Indica el tipo de MD que se ha registrado. Puede variar entre 4 tipos: termómetro, báscula, medidor de presión arterial o pulsioxímetro.

- **ProtocolType (1.3.6.1.4.1.28308.4.2.1.1.6):** Indica el tipo de protocolo utilizado en la comunicación entre MD y manager. En este caso será por defecto el protocolo X.73, pero puede variar dependiendo del sistema en el que nos encontremos.

- **ConfigurationType (1.3.6.1.4.1.28308.4.2.1.1.7):** Muestra el tipo de configuración con la que está trabajando el MD. Puede ser o estándar o extendida.

- **UpdateRequestState (1.3.6.1.4.1.28308.4.2.1.1.8):** Este parámetro se utiliza para pedir actualizaciones del estado de éste dispositivo en concreto.

- **UpdateRequestTechnicalData (1.3.6.1.4.1.28308.4.2.1.1.9):** Este parámetro se utiliza para pedir actualizaciones del estado de éste dispositivo en concreto.

- **DeviceDate (1.3.6.1.4.1.28308.4.2.1.1.10):** Indica la fecha en que se registró este MD por primera vez en el CE.

C.2.2 MedicalDeviceDataTable

Los objetos de esta tabla se describen a continuación:

- **IdMDData (1.3.6.1.4.1.28308.4.2.2.1.1):** Identificador del MD al que corresponden los datos que se registran en esta entrada.

- **IdCapture (1.3.6.1.4.1.28308.4.2.2.1.2):** Identificador de la captura que nos indica el número de medidas que se han tomado de ese MD en concreto hasta ese momento.

- **SupplyType (1.3.6.1.4.1.28308.4.2.2.1.3):** Indica el tipo de alimentación que está utilizando el MD, que puede variar entre batería o conexión a la red eléctrica.

- **BatteryLevel (1.3.6.1.4.1.28308.4.2.2.1.4):** Muestra en tanto por cierto la cantidad de batería que se queda al MD.

- **RemainderBattery (1.3.6.1.4.1.28308.4.2.2.1.5):** Indica el número estimado de horas que puede funcionar el MD con la batería que le queda.

- **Errors (1.3.6.1.4.1.28308.4.2.2.1.6):** Muestra si ha habido algún error en el MD al tomar esta medida. Su valor es "OK" si no ha habido problema, y "Error" si lo ha habido.

- **OwnerMD (1.3.6.1.4.1.28308.4.2.2.1.7):** Indica la persona que ha realizado la medida. Este objeto puede servir para los casos en que más de una persona utilice el MD.

- **DateTimeData (1.3.6.1.4.1.28308.4.2.2.1.8):** Muestra la fecha en la que se realizó esta entrada en la tabla.

C.2.3 MedicalDeviceStateTable

La descripción de los objetos que se pueden observar en esta tabla es la siguiente:

- **IdMDState (1.3.6.1.4.1.28308.4.2.3.1.1):** Identificador del MD al que corresponde el estado que se registra en esta entrada.

- **IdState (1.3.6.1.4.1.28308.4.2.3.1.2):** Identificador del estado dentro de los presentes para este MD.

- **DeviceState (1.3.6.1.4.1.28308.4.2.3.1.3):** Valor del nuevo estado del MD.Este valor será una de estas posibilidades: *available*, *notavailable*, *disconnected*, *connected*, *associated* u *operating*.

- **DateTimeState (1.3.6.1.4.1.28308.4.2.3.1.4):** Muestra la fecha en la que se realizó esta entrada en la tabla.

C.2.4 SpecificErrorsTable

En la figura **??** se muestran los objetos detallados a continuación:

- **IdMDError (1.3.6.1.4.1.28308.4.2.4.1.1):** Identificador del MD al que corresponde el error que se registra en esta entrada.

- **IdError (1.3.6.1.4.1.28308.4.2.4.1.2):** Identificador del error dentro de los presentes para este MD.

- **DeviceWorking (1.3.6.1.4.1.28308.4.2.4.1.3):** Indica si hay algún error en el funcionamiento general del MD. En caso de haberlo, si valor será "Error"; si no lo hay, el valor será "OK".

- **SensorWorking (1.3.6.1.4.1.28308.4.2.4.1.4):** Indica si hay algún error en el funcionamiento del sensor del MD, en caso de que posea sensor. En caso de haberlo, si valor será "Error"; si no lo hay, el valor será "OK".

- **DeviceConnection (1.3.6.1.4.1.28308.4.2.4.1.5):** Indica si hay algún error en la conexión del MD con el CE. En caso de haberlo, si valor será "Error"; si no lo hay, el valor será "OK".

- **SensorJamming (1.3.6.1.4.1.28308.4.2.4.1.6):** Indica si hay algún error debido a interferencias en el sensor del MD, en caso de poseer sensor. En caso de haberlo, si valor será "Error"; si no lo hay, el valor será "OK".

- **SignalFailures (1.3.6.1.4.1.28308.4.2.4.1.7):** Indica si hay algún error debido a que las señales médicas que utiliza el dispositivo para calcular los valores médicos son pobres o tienen algún problema. En caso de haberlo, si valor será "Error"; si no lo hay, el valor será "OK".

C.3 Grupo AlarmTable

La descripción de los objetos pertenecientes a esta tabla es la siguiente:

- **IdDevice (1.3.6.1.4.1.28308.4.3.1.1):** Identificador del MD para el cual se ha definido esta alarma.

- **IdAlarm (1.3.6.1.4.1.28308.4.3.1.2):** Identificador de la alarma. Este identificador no tiene por qué ser consecutivo, el gestor que crea la alarma elige su valor.

- **OidVariable (1.3.6.1.4.1.28308.4.3.1.3):** Muestra el OID que vamos a monitorizar para comprobar que sus valores no hacen activarse a la alarma.

- **IdEventAlarmValue (1.3.6.1.4.1.28308.4.3.1.4):** Identificador del evento asociado a esta alarma cuando la variable que monitorizamos alcanza el valor indicado en AlarmValue.

- **AlarmValue (1.3.6.1.4.1.28308.4.3.1.5):** Cuando la variable con OID igual al del campo OidVariable alcanza el valor indicado en este objeto, se activa la alarma.

- **IdEventUp (1.3.6.1.4.1.28308.4.3.1.6):** Identificador del evento asociado a esta alarma cuando la variable que monitorizamos sobrepasa por encima el valor indicado por ThresholdUp.

- **IdEventDown (1.3.6.1.4.1.28308.4.3.1.7):** Identificador del evento asociado a esta alarma cuando la variable que monitorizamos sobrepasa por debajo el valor indicado por ThresholdDown.

- **ThresholdUp (1.3.6.1.4.1.28308.4.3.1.8):** Cuando la variable con OID igual al del campo OidVariable sobrepasa por encima el valor indicado en este objeto, se activa la alarma.

- **ThresholdUp (1.3.6.1.4.1.28308.4.3.1.8):** Cuando la variable con OID igual al del campo OidVariable sobrepasa por debajo el valor indicado en este objeto, se activa la alarma.

- **OwnerAlarm (1.3.6.1.4.1.28308.4.3.1.9):** Muestra el identificador del gestor que ha creado la alarma.

- **StatusAlarm (1.3.6.1.4.1.28308.4.3.1.10):** Muestra el estado de creación de la alarma. Si el valor es 1 (valid), implica que la entrada está creada y completada. Si el valor es 3 (underCreation), implica que la entrada aún está por completar. Los valores 2(creation) y 4 (drop) sirven para crear y borrar la entrada, respectivamente.

C.4 Grupo EventTables

C.4.1 ConfigEventTable

La descripción de los objetos visibles en esta tabla es la siguiente:

- **IdEvent (1.3.6.1.4.1.28308.4.4.1.1.1):** Identificador del evento. El gestor puede darle el valor que quiera a este objeto al configurar el evento.

- **EventType (1.3.6.1.4.1.28308.4.4.1.1.2):** Indica el tipo de evento que se ha creado. El tipo de evento describe las acciones que se realizan al activar la alarma. Estas acciones pueden variar entre enviar un trap, crear una entrada en la tabla de logs o ambas acciones.

- **trapType (1.3.6.1.4.1.28308.4.4.1.1.3):** Indica el tipo de trap que se va a enviar al gestor. Los tipos de trap son: SystemOvercharged, WrongDeviceWorking, SpecificError, NewMD y Warning.

- **OwnerEvent (1.3.6.1.4.1.28308.4.4.1.1.4):** Este objeto muestra el identificador del gestor que configuró este evento.

- **StatusEvent (1.3.6.1.4.1.28308.4.4.1.1.5):** Muestra el estado de creación del evento. Si el valor es 1 (valid), implica que la entrada está creada y completada. Si el valor es 3 (underCreation), implica que la entrada aún está por completar. Los valores 2(creation) y 4 (drop) sirven para crear y borrar la entrada, respectivamente.

C.4.2 LogTable

A continuación se describen los objetos de la tabla:

- **IdEventLog (1.3.6.1.4.1.28308.4.4.2.1.1):** Identificador del evento que responsable de esta entrada en la tabla de logs.
- **IdLog (1.3.6.1.4.1.28308.4.4.2.1.2):** Identificador del log.
- **IdAlarm (1.3.6.1.4.1.28308.4.4.2.1.3):** Identificador de la alarma responsable de la activación del evento y por tanto de la creación del log.
- **LogDescription (1.3.6.1.4.1.28308.4.4.2.1.4):** Describe brevemente la causa de la activación de la alarma.
- **DateTimeLog (1.3.6.1.4.1.28308.4.4.2.1.5):** Indica la fecha en la que fué creado este log.

C.5 Grupo ManagerTable

La descripción de este último grupo es la que sigue:

- **IdManager (1.3.6.1.4.1.28308.4.5.1.1):** Identificador del gestor.
- **Owner (1.3.6.1.4.1.28308.4.5.1.2):** Muestra el nombre del gestor.
- **IP (1.3.6.1.4.1.28308.4.5.1.3):** Indica la IP del gestor para poder enviarle los Traps que sean necesarios.
- **PhoneNumber (1.3.6.1.4.1.28308.4.5.1.4):** Muestra un teléfono de contacto para contactar con el gestor en caso de ser necesario.
- **Email (1.3.6.1.4.1.28308.4.5.1.5):** Contiene un email para poder contactar con el gestor por correo electrónico.
- **Workingplace (1.3.6.1.4.1.28308.4.5.1.6):** Indica una dirección donde se puede comunicar con el gestor de manera física.

- **Permission (1.3.6.1.4.1.28308.4.5.1.7):** Indica los privilegios del gestor dentro de la MIB.

- **NextFreeIndex (1.3.6.1.4.1.28308.4.5.1.8):** Indica cuál es el índice que se debe asignar al siguiente gestor que se registre en la tabla, ya que en esta tabla los índices son consecutivos.

- **StatusManager (1.3.6.1.4.1.28308.4.5.1.9):** Muestra el estado de creación del gestor. Si el valor es 1 (valid), implica que la entrada está creada y completada. Si el valor es 3 (underCreation), implica que la entrada aún está por completar. Los valores 2(creation) y 4 (drop) sirven para crear y borrar la entrada, respectivamente.

Anexo D

Estructura interna de la MIB MedicasDevicesManager

D.1 Estructura interna de tablas para una MIB genérica

Para realizar la organización de la base de datos en la que se implementan las MIBs se propone un esquema basado en la utilización de distintos tipos de tablas. De esta forma conseguiremos un diseño que permita la implementación de una MIB genérica independiente del soporte físico utilizado. El diseño propuesto permitirá además mantener la estructura en árbol de la MIB. La estructura interna propuesta se basa en la utilización de tres tipos de tablas distintas para almacenar los objetos de una MIB genérica de forma ordenada: "tablaInicial" , "tablaControl" y "tabla de Instancias". A continuación se describe la organización de las mismas y se muestra un ejemplo de su utilización realizando la organización interna de la MIB if (grupo de MIBs perteneciente a las MIB2 que contiene información de tráfico asociado a cada uno de los interfaces disponibles en un dispositivo).

D.1.1 Estructura genérica de una tabla inicial

En la tabla incial tendremos la información de los objetos, tablas o grupos que integran en un primer nivel la MIB estudiada. En la tabla D.1 se observa la

estructura de esta tabla, el contenido de sus campos y la descripción de los mismos.

OID	Status	Name	Next_Name	Next_OID	Type_value	Value

Tabla D.1. Definición de tabla inicial.

1. OID: OID que identifica a cada uno de los objetos o grupos que se encuentran en el primer nivel de la estructura de la MIB.

2. Status: Indica la accesibilidad a los valores contenidos en el grupo identificado por el OID.

   - 0: Valor No accesible, porque el OID identifica a una rama.
   - 1: Valor accesible. El OID corresponde a un objeto a cuyo valor sólo se puede acceder en modo de lectura.
   - 2: Valor accesible. El OID identifica a un objeto cuyo valor es accesible tanto en modo lectura como en modo escritura.
   - 3: Valor no accesible en esta tabla porque el OID identifica a una tabla o a un grupo de objetos relacionados, pero a los que va a ser posible acceder gracias a la creación de tablas de control e instancias, en las que se definen y estructura su contenido.

3. Name: Nombre de la tabla en la que se guardan todas las entradas si el OID apunta a una tabla o a un grupo. El campo *Name* contendrá el valor de la propia tabla si el OID corresponde a un objeto accesible directamente en la misma tabla.

4. Next_Name: Nombre de la tabla en la que se almacena el objeto siguiente. Si el siguiente objeto se encuentra en la misma tabla, aparecerá en next_name el nombre de la tabla en la que se lanzan las sentencias SQL,es decir, el mismo nombre de la tabla.

5. Next_OID: OID del objeto siguiente.

6. Type_value: Tipo de los valores almacenados en caso de que el OID represente a un objeto (string ,timeticks, int...).Para las entradas que no identifiquen objetos este valor será igual a 0.

7. Value: Valor asociado al objeto. *Value* tomará el valor 0 por defecto para los casos en los que el OID no identifique a un objeto, aunque lógicamente, nunca llegará a leerse, por estar indicado en el status que ese es un valor no accesible.

Las ramas que identifiquen a un grupo o a una tabla, indexarán otras tablas en las que se estructurará el contenido para el acceso de las mismas de forma similar a la tabla de inicio. Si el contenido de la MIB interfaces (contenida en la MIB-II) tuviera una organización interna en base de datos como la propuesta, su tabla de inicio sería la representada por la tabla D.2. El OID 1.3.6.1.2.1.2 identifica al grupo de objetos almacenados en la MIB interfaces, el OID 1.3.6.1.2.1.2.1.0 al objeto 1(sólo se almacena en la tabla el OID asociado al nivel de profundidad de la MIB, pero el acceso a la misma deberá hacerse indexando a este el .0 necesario para acceder a un objeto :1.3.6.1.2.1.2.1.0) ,cuyo valor es sólo de lectura y al que podemos acceder directamente en esta tabla de inicio, y por el último el OID 1.3.6.1.2.1.2.2 indica que representa a una tabla cuya organización está especificada en la tabla de nombre *iftable*. El *next_name* del objeto identificado por el OID 1.3.6.1.2.1.2.1.0 será el primer objeto accesible perteneciente a la tabla *Iftable*, es decir, el objeto *IfIndex* correspondiente al primer interfaz de los dos de los que dispone el host. Con esta organización, disponemos de la información suficiente para poder hacer un *getnext* del OID 1.3.6.1.2.1.2.1.0 y acceder de forma inmediata a la tabla en la que se almacena el primer objeto encontrado con OID siguiente al especificado.

OID	Status	Name	Next_Name	Next_OID	Type_value	Value
1.3.6.1.2.1.2	0	branch	TablaInicial	1.3.6.1.2.1.2.1	0	0
1.3.6.1.2.1.2.1	1	TablaInicial	ifIndex	1.3.6.1.2.1.2.2.1.1	int	2
1.3.6.1.2.1.2.2	3	ifIndex	atIfIndex	1.3.6.1.2.1.3.1.1.1	0	0

Tabla D.2. Ejemplo de tabla inicial: MIB interfaces.

D.1.2 Estructura genérica de una tabla de control

Una vez inicializada la tabla de inicio manteniendo la estructura de ramas de la MIB, se inicializarán todas las tablas y grupos definidas en la tabla anterior. De esta forma, una vez que el agente acceda a la tabla de inicio y averigüe a que tipo de tabla, objeto, o grupo, pertenece el OID preguntado, lanzará de nuevo una sentencia de búsqueda sobre la tabla o grupo cuyo nombre se especificaba en la tabla de inicio, en caso de no corresponder a un objeto de la tabla anterior el OID dado.

Se propone la utilización de una "tablaControl" que señale el nombre de la tabla en la que se almacenan todas las entradas pertenecientes a dicha tabla encuestada, correspondientes a un campo concreto, y en la que se especifica además el *next_oid* en caso de realizar una petición de *getnext* sobre algún identificador de los que contiene. La organización del contenido de las tablas se plantea de esta forma para poder agrupar de forma más ordenada todas las entradas y poder implementar el comando *getnext* de forma más clara y sencilla. Esta estructura permite tener una organización de la tabla independiente del número de instancias que se tengan, es por ello, que el OID que aparece en esta tabla es el OID genérico que identifica al campo y no a una instancia de la tabla en particular. La estructura genérica de la tabla Control y el contenido y descripción de sus campos se muestran en la tabla D.3.

OID	Type	Status	Entrances	Name	Next_Name	Next_OID

Tabla D.3. Definición de tabla control.

1. OID: OID que identifica al campo dentro de la tabla.

2. Type: Tipo de valor que se va a almacenar en las tablas correspondientes a cada OID.(Enteros, cadenas de caracteres...).

3. Status: Define la accesibilidad de los campos. Existen tres posibles valores para el status: 0, 1 y 2.

- 0:Valor No accesible. El OID no estará representando a un objeto.
- 1:Sólo lectura.
- 2:Lectura Escritura.

4. Entrances: Número de entradas existentes en la tabla.

5. Name: Nombre de la tabla en la que se almacenan los valores correspondientes a ese OID.

6. Next_Name: Nombre de la siguiente variable dentro de esta misma tabla.

7. Next_OID: OID que identifica al objeto siguiente. Este campo será de utilidad a la hora de programar el *getnext*.

Asociadas a cada uno de los OID que identifican a un objeto se crearán tablas, cuyos nombres se describen en el campo *Name* de la tabla de control, y que almacenarán para ese identificador concreto todas las entradas correspondientes a distintos usuarios gestionados.

Siguiendo con el ejemplo descrito para la tabla de inicio, en la tabla D.4 se describe cuál sería la estructura de la "tablacontrol" para la tabla *iftable* (contenida en el grupo if de las MIB-II) que contiene dos entradas en su tabla de interfaces. Como se observa en el ejemplo, se propone utilizar el mismo nombre que identifica al campo para nombrar la tabla que contiene los objetos que describe.

OID	Type	Sta.	ent.	Name	Next_Name	Next_OID
1.3.6.1.2.1.2.2.1	null	0	2	branch	ifIndex	1.3.6.1.2.1.2.2.1.1.1
1.3.6.1.2.1.2.2.1.1	OctetString	1	2	ifIndex	ifType	1.3.6.1.2.1.2.2.1.1.1
1.3.6.1.2.1.2.2.1.2	integer	1	2	ifType	ifMtu	1.3.6.1.2.1.2.2.1.2.1
1.3.6.1.2.1.2.2.1.3	integer	1	2	ifMtu	ifSpeed	1.3.6.1.2.1.2.2.1.3.1
...	...	...	...	...	...	...
1.3.6.1.2.1.2.2.1.22	ObjectID	1	2	ifSpecific	atIfIndex	1.3.6.1.2.1.2.2.1.22.1

Tabla D.4. Ejemplo de tabla de control: tabla ifTable.

D.1.3 Estructura genérica de una tabla de instancias

D.1.3.1 Tabla de instancias de control

Para la definición de la estructura interna de una MIB genérica, se definen dos tipos distintos de tablas de instancias, unas asociadas a las tablas de datos y de estados de los MDs, y el otro tipo asociado a todas las demás. Esto es debido a que apra los datos y los estados debemos controlar la posición de la medida. La estructura genérica en la que se almacenan las instancias de uno de los objetos que contenían las tablas de control es la mostrada en la tabla D.5.

OID	Índice1	...	Índicen	Value	Position	Next_OID

Tabla D.5. Definición de la estructura de una tabla de instancias de control.

1. OID: OID que identifica de forma unívoca a una entrada y campo de la tabla.

2. Indice: Conjunto de índices que identifican a cada una de las instancias y sirven para indexar la tabla. Estos valores forman parte del OID especificado en el campo anterior. La utilización de estos índices permite una óptima programación del comando *getnext*.

3. Value: Valor del objeto asociado.

4. Next_OID: OID que identifica al objeto siguiente.

En el ejemplo utilizado sólo existe un campo para los índices puesto que sólo hay un índice para indexar esta tabla. En la tabla D.6 se muestra la tabla de instancias del campo *Ifspeed*.

OID	Índice	Value	Position	Next_OID
1.3.6.1.2.1.2.2.1.5.1	1	10000000	1	1.3.6.1.2.1.2.2.1.5.2
1.3.6.1.2.1.2.2.1.5.2	2	100000000	2	1.3.6.1.2.1.2.2.1.6.1

Tabla D.6. Ejemplo de tabla de instancias: tabla ifSpeed.

D.1.3.2 Tabla de instancias de datos

Su estructura es semejante a la tabla de instancias de parámetros. En este caso se añade un campo llamado Visible, el cual determina la visibilidad de la observación. Puesto que al crear la primera entrada de un dispositivo en estas tablas se crean 20 entradas a la vez, este campo nos permite saber cuáles de esas entradas tienen un dato válido y cuales están vacías (0:no visible,1:Visible). Mediante el campo Position podemos saber cuál es el orden de las medidas de cada dispositivo, dado que cuando se llenen las 20 entradas disponibles habrá que ir actualizando las entradas más antiguas por otras más nuevas y los datos cambiarán de posición dentro de la misma tabla. En la tabla D.7 se muestra la estructura mencionada.

OID	Índice1	...	Índicen	Value	Visible	Position	Next_OID

Tabla D.7. Definición de la estructura de una tabla de instancias de datos.

D.2 MySQL

MySQL es el Sistema de Gestión de Bases de Datos (*Database Management System*,DBMS, en inglés) de código abierto más popular en la actualidad. Trabaja con bases de datos relacionales cumpliendo el Standard ANSI/ISO SQL(*Structured Query Language*). La utilización de esta base de datos va a permitir, entre otras muchas ventajas, realizar accesos de forma muy rápida, conseguir estabilidad en el manejo de múltiples conexiones simultáneas, obtener seguridad en forma de permisos y privilegios, conectividad, es multihilo, con lo que puede beneficiarse de sistemas multiprocesador, está escrito en C y C++ y probado con multitud de compiladores, etc.

El acceso a la base de datos, se realiza a través de un motor que hace las funciones de intérprete ahorrando una gran cantidad de trabajo. Además, la utilización de sentencias SQL (lenguaje muy potente para la consulta de base de datos) para la interacción entre la base de datos y el agente, proporciona

portabilidad al diseño. Al ser SQL un lenguaje estandarizado, las consultas hechas realizando este lenguaje son fácilmente portables a otros sistemas y plataformas. Las cualidades destacadas, su gran facilidad de uso y la existencia de herramientas como phpMyAdmin, que permite administrar las bases de datos MYSQL mediante una interfaz sencilla creada en PHP (útiles para la programación del sistema) hacen que MYSQL presente las suficientes cualidades como para ser la opción más atractiva para la implementación de la MIB. MYSQL proporciona conectividad a las aplicaciones cliente desarrolladas en Java mediante un driver JDBC.(Java Database Connectivity) denominado MySQL Connector/J.

D.3 Organización interna de la MIB Medical Devices Manager

Inicialmente en la base de datos disponemos de 11 tablas (una tabla incial y una más por cada tabla existente en la MIB), que se crean al iniciar el agente y que engloban todos los campos presentes en la MIB. Al inicializar las tablas establecemos un valor nulo para el *Next_OID*, que llamaremos "EndOfMIB", para representar que las tablas están vacías y por tanto no hay un OID posterior en la MIB con valor. Una vez se van rellenando las tablas, se crean nuevas tablas para las instancias y se actualizan los *Next_OIDs* que correspondan. El campo *Next_Name* apunta al siguiente objeto de la MIB. Esto nos va a ayudar a responder a las peticiones del gestor tipo *GetBulk* de forma correcta.

La tabla inicial y la tabla correspondiente a la CEControlTable de la MIB tendrán una estructura del tipo tabla inicial anteriormente descrito. Como ejemplo vemos la tabla de control del CE en la tabla D.8. Para facilitar la visión de los campos *OID* y *Next_OID*, se omite la parte inicial 1.3.6.1.4.1., de modo que solo vemos la parte del identificador a partir del grupo zaragozaNetworkManagementResearchGroup. Es decir, el OID 1.3.6.1.4.1.28308.4.1.1 se verá en las tablas como 28308.4.1.1.

Para el resto de las tablas que se crean al iniciar el agente utilizamos el esquema

OID	Status	Name	Next_Name	Next_OID	Type	Value
28308.4.1.1	1	IdComputeEngine	DeviceType	EndOfMib	OctetString	0
28308.4.1.2	1	DeviceType	WorkingState	EndOfMib	OctetString	0
28308.4.1.3	1	WorkingState	CommunicationState	EndOfMib	OctetString	0
28308.4.1.4	1	CommunicationState	AsociatedMDs	EndOfMib	OctetString	0
28308.4.1.5	1	AsociatedMDs	BandwithUse	EndOfMib	Integer	0
28308.4.1.6	1	BandwithUse	CpuUse	EndOfMib	Integer	0
28308.4.1.7	1	CpuUse	HardDiskMemoryUse	EndOfMib	Integer	0
28308.4.1.8	1	HardDiskMemoryUse	VirtualMemoryUse	EndOfMib	Integer	0
28308.4.1.9	1	VirtualMemoryUse	PhysicalMemoryUse	EndOfMib	Integer	0
28308.4.1.10	1	PhysicalMemoryUse	UserContactInfo	EndOfMib	Integer	0
28308.4.1.11	1	UserContactInfo	HostIPAddress	EndOfMib	OctetString	0
28308.4.1.12	1	HostIPAddress	UpdateRequestStateMDs	EndOfMib	IpAddres	0
28308.4.1.13	2	UpdateRequestStateMDs	UpdateRequestTechnicalDataMDs	EndOfMib	Integer	0
28308.4.1.14	2	UpdateRequestTechnicalDataMDs	DateTimeControl	EndOfMib	Integer	0
28308.4.1.15	1	DateTimeControl	IdMDControl	EndOfMib	Date	0

Tabla D.8. Tabla de MySQL que controla la tabla CEControlTable.

de las tablas de control. En la tabla D.9 vemos un ejemplo de una de estas tablas.

OID	Type	Status	Entrances	Name	Next_Name	Next_OID
28308.4.2.1.1	Null	0	0	branch	IdMDControl	EndOfMib
28308.4.2.1.1.1	OctetString	1	0	IdMDControl	Manufacturer	EndOfMib
28308.4.2.1.1.2	OctetString	1	0	Manufacturer	Model	EndOfMib
28308.4.2.1.1.3	OctetString	1	0	Model	SystemId	EndOfMib
28308.4.2.1.1.4	OctetString	1	0	SystemId	MDType	EndOfMib
28308.4.2.1.1.5	OctetString	1	0	MDType	ProtocolType	EndOfMib
28308.4.2.1.1.6	OctetString	1	0	ProtocolType	ConfigurationType	EndOfMib
28308.4.2.1.1.7	OctetString	1	0	ConfigurationType	UpdateRequestState	EndOfMib
28308.4.2.1.1.8	OctetString	2	0	UpdateResquestState	UpdateRequestTechnicalData	EndOfMib
28308.4.2.1.1.9	OctetString	2	0	UpdateRequestTechnicalData	DeviceDate	EndOfMib
28308.4.2.1.1.10	OctetString	1	0	DeviceDate	IdMDState	EndOfMib

Tabla D.9. Tabla de MySQL que controla la tabla MDControlTable.

A medida que se van recibiendo datos de los dispositivos médicos, y que los gestores configuran alarmas y eventos, se crean tablas de instancias. En concreto, cada vez que un objeto de la MIB, presentes todos ellos en las tablas de control, recibe un valor y todavía no tenía ninguno anterior, se crea una tabla de instancias para ese objeto, que será de uno de los dos tipos descritos en el apartado anterior, es decir, de control o de datos, dependiendo de la tabla a la que pertenezca dicho objeto. Los objetos pertenecientes a las tablas de datos y estados tendrán una tabla de instancias de datos, similar a la que observamos en la tabla D.10.

En el resto de objetos nos encontramos tablas de instancias de tipo control. Un ejemplo ilustrativo es la tabla D.11.

Los índices que se utilizan para indexar las tablas de instancias no depeden de si se trata de una tabla de instancias de control o de datos, sino de la tabla en al que nos encontramos. Como se describió en la sección 3.2, el identificador estará formado por el OID del objeto de la MIB al que correspondan, añadiendo

OID	Indice	IndiceObs	Visible	Posicion	Valor	Next_OID
28308.4.2.3.1.2.1.1	1	1	1	1	AVAILABLE	28308.4.2.3.1.2.1.2
1.3.6.1.4.1.28308.4.2.3.1.2.1.2	1	2	1	2	CONNECTED	28308.4.2.3.1.2.1.3
28308.4.2.3.1.2.1.3	1	3	1	3	ASOCIATED	28308.4.2.3.1.2.3.1
28308.4.2.3.1.2.1.4	1	4	0	4		28308.4.2.3.1.2.3.1
...	...	...	...	...	...	...
28308.4.2.3.1.2.1.20	1	20	0	20		28308.4.2.3.1.2.3.1
28308.4.2.3.1.2.3.1	3	1	1	1	AVAILABLE	28308.4.2.3.1.3.1.1
28308.4.2.3.1.2.3.2	3	2	0	2		28308.4.2.3.1.3.1.1

Tabla D.10. Ejemplo de una tabla de instancias de datos para el objeto IdState.

OID	Indice	Valor	Next_OID
28308.4.2.1.1.4.1	1	12345	28308.4.2.1.1.4.2
28308.4.2.1.1.4.2	2	11111	28308.4.2.1.1.4.3
28308.4.2.1.1.4.3	3	42657	28308.4.2.1.1.5.1

Tabla D.11. Ejemplo de tabla de instancias de control para el objeto SystemId.

al final el índice del dispositivo, manager o evento al que pertenece la entrada, y en algunos casos se le añadirá un índice que representa la posición de la entrada correspondiente dentro de las existentes para ese dispositivo o evento.

Anexo E

Definición formal de la MIB Medical Devices Manager

```
-- Definition of the tables used to manage the medical devices
-- and the computer engine
-- File: ???
-- Changes: none needed
--
-- Andreas Muñoz Zuara : 2-11-2010

MEDICALDEVICESMANAGER-MIB DEFINITIONS ::= BEGIN
IMPORTS

MODULE-IDENTITY, OBJECT-TYPE, OBJECT-IDENTITY,
NOTIFICATION-TYPE, enterprises, Counter32,
Integer32,IpAddress, TimeTicks FROM SNMPv2-SMI

TEXTUAL-CONVENTION, DisplayString FROM SNMPv2-TC

MODULE-COMPLIANCE, OBJECT-GROUP, NOTIFICATION-GROUP FROM SNMPv2-CONF;
```

```
medicalDevicesManager MODULE-IDENTITY

LAST-UPDATED ‘‘200005110000Z’’ -- 2 Nov, 2010
ORGANIZATION ‘‘Unizar’’
CONTACT-INFO
‘‘Andreas Muñoz Zuara
Phone: +1-650-948-6500
Fax: +1-650-745-0671
Email: andreasmunozzuara@gmail.com ’’

DESCRIPTION
‘‘This MIB defines objects for managing of medical devices and
compute engines. The MedicalDevicesManager MIB allows us to check
the technical parameters of the devices, add new devices to the
computer engine and inform the responsibles of the correct working
of these machines when some of the parameters exceeds a threshold.’’

REVISION ‘‘1’’ --2010
DESCRIPTION
‘‘The original version of this MIB.’’
::= {zaragozaNetworkManagementResearchGroup 4}

zaragozaNetworkManagementResearchGroup
OBJECT IDENTIFIER ::= { enterprises 28308 }


medicalDevicesManager
OBJECT IDENTIFIER ::= {zaragozaNetworkManagementResearchGroup 4}
```

```
OwnerString ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
‘‘This data type is used to model an administratively assigned name
of the owner of a resource. Implementations must accept values
composed of well-formed NVT ASCII sequences. In addition,
implementations shouldaccept values composed of well-formed UTF-8
sequences. SNMP access control is articulated entirely in terms
of the contents of MIB views;access to a particular SNMP object
instance depends only upon its presence or absence in a particular
MIB view and never upon its value or the value of related object
instances. Thus, objects of this type afford resolution of resource
contention only among cooperating managers; they realize no access
control function with respect to uncooperative parties.’’
SYNTAX OCTET STRING (SIZE (0..127))

DateTime ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
‘‘This data type is used to defined a date as dd/mm/aa_hh:mm:seg’’
SYNTAX OCTET STRING (SIZE (0..127))

DeviceStatus ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
‘‘This data type defines the amount of different posible states of
working of the devices.’’
SYNTAX INTEGER {
Available (1),
NotAvailable (2),
Disconnected (3),
```

```
Connected (4),
Associated (5),
Operating (6),
}

PossibleTraps ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
‘‘This data type defines  the different traps that the agent can
throw to the managers.’’
SYNTAX INTEGER {
systemOvercharged (1),
wrongDeviceWorking (2),
specificError (3),
newMD (4),
warning (5)
}

EntryStatus ::= TEXTUAL-CONVENTION
STATUS current
DESCRIPTION
‘‘The status of a table entry.

Setting this object to the value invalid(4) has the effect of
invalidating the corresponding entry.That is, it effectively
disassociates the mapping identified with said entry. It is an
implementation-specific matter as to whether the agent removes an
invalidated entry from the table. Accordingly, management stations
must be prepared to receive tabular information from agents that
corresponds to entries currently not in use. Proper interpretation
of such entries requires examination of the relevant EntryStatus
```

object. An existing instance of this object cannot be set to createRequest(2). This object may only be set to createRequest(2) when this instance is created. When this object is created, the agentmay wish to create supplemental object instances with default values to complete a conceptual row in this table. Because the creation of these default objects is entirely at the option of the agent, the manager must not assume that any will be created, but may make use of any that are created. Immediately after completing the create operation, the agent must set this object to underCreation(3). When in the underCreation(3) state, an entry is allowed to exist in a possibly incomplete, possibly inconsistent state, usually to allow it to be modified in multiple PDUs. When in this state, an entry is not fully active. Entries shall exist in the underCreation(3) state until the management station is finished configuring the entry and sets this object to valid(1) or aborts, setting this objectto invalid(4). If the agent determines that an entry has been in the underCreation(3) state for an abnormally long time, it may decidethat the management station has crashed. If the agent makes this decision, it may set this object to invalid(4) to reclaim the entry. A prudent agent will understand that the management station may need to wait for human input and will allow for that possibility in its determination of this abnormally long period.An entry in the valid(1) state is fully configured and consistent and fully represents the configuration or operation such a row is intended to represent. For example, it could be a statistical function that is configured and active, or a filter that is available in the list of filters processed by the packet capture process. A manager is restricted to changing the state of an entry in the following ways:

To: valid createRequest underCreation invalid

From:
valid OK NO NO OK
createRequest N/A N/A N/A N/A
underCreation OK NO OK OK
invalid NO NO NO OK
nonExistent NO OK NO NO

In the table above, it is not applicable to move the state from the createRequest state to any other state because the manager will never find the variable in that state. The nonExistent state is not a value of the enumeration, rather it means that the entryStatus variable does not exist at all. An agent may allow an entryStatus variable to change state in additional ways, so long as the semantics of the states are followed. This allowance is made to ease the implementation of the agent and is made despite the fact that managers should never exercise these additional state transitions.''
SYNTAX INTEGER {
valid(1),
createRequest(2),
underCreation(3),
invalid(4)
}

computeEngineControlInfo OBJECT IDENTIFIER
::={medicalDevicesManager 1}

idComputeEngine OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only

```
STATUS mandatory
DESCRIPTION ‘‘This is the identifier for this compute engine
inside the network.’’
::= { computeEngineControlInfo 1}

deviceType OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This object contains the type of compute engine we
are monitorising. It could be a PDA, a PC, etc. ’’
::= { computeEngineControlInfo 2}

workingState OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This indicates if the CE is working properly or if
there are some troubles. It has two states:
operative and damaged. ’’
::= { computeEngineControlInfo 3}

communicationState OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This indicates if the CE is sending, receiving or
idle. ’’
::= { computeEngineControlInfo 4}

asociatedMDs OBJECT-TYPE
```

```
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This indicates the number of MDs connected to
this CE. ’’
::= { computeEngineControlInfo 5}

bandwidthUse OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This shows the percentage of use of the total bandwidth
of the CE. ’’
::= { computeEngineControlInfo 6}

cpuUse OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This shows the percentage of use of the total capacity
of the CPU. ’’
::= { computeEngineControlInfo 7}

hardDiskMemoryUse OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This shows the percentage of use of the hard disk
memory of the CE.’’
::= { computeEngineControlInfo 8}
```

```
virtualMemoryUse OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This shows the percentage of use of the virtual
memory of the CE.’’
::= { computeEngineControlInfo 9}

physicalMemoryUse OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This shows the percentage of use of the physical
memory of the CE.’’
::= { computeEngineControlInfo 10}

userContactInformation OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This keeps information about the user of the CE to
contact him in case it is needed. ’’
::= { computeEngineControlInfo 11}

hostIPAddress OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘It is the IP of this compute engine. ’’
::= { computeEngineControlInfo 12}
```

```
updateRequestStateMDs OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is used by the manager to update the state of
all the devices connected to the CE. ’’
::= { computeEngineControlInfo 13}

updateRequestTechnicalDataMDs OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is used by the manager to update the technical
information of all the devices connected to the CE. ’’
::= { computeEngineControlInfo 14}

dateTimeControl OBJECT-TYPE
SYNTAX DateTime
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This indicates when was the last update of the entry
on the table. ’’
::= { computeEngineControlInfo 15}


medicalDeviceInfo OBJECT IDENTIFIER ::= {medicalDevicesManager 2}

medicalDeviceControlTable OBJECT-TYPE
SYNTAX SEQUENCE OF MedicalDeviceControlEntry
ACCESS not-accessible
STATUS mandatory
```

```
DESCRIPTION ‘‘This table can store control parameters of the
medical devices.’’
::= {medicalDeviceInfo 1}

medicalDeviceControlEntry OBJECT-TYPE
SYNTAX  MedicalDeviceControlEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the control parameters of each
medical device.’’
INDEX {idMDControl}
::= {medicalDeviceControlTable 1}

MedicalDeviceControlEntry ::= SEQUENCE {
idMDControl   OCTET STRING,
manufacturer OCTET STRING,
model OCTET STRING,
systemId OCTET STRING,
mdType OCTET STRING,
protocolType OCTET STRING,
configurationType OCTET STRING,
updateRequestState Integer32,
updateRequestTechnicalData Integer32,
deviceDate DateTime
}

idMDControl OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the unique identifier of each medical
```

```
device connected to the compute engine.''
::= { medicalDeviceControlEntry 1}

manufacturer OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ``This object give us the name of the manufacturer
of the medical device.''
::= { medicalDeviceControlEntry 2}

model OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ``This is the model of the medical device inside
the range of devices of the same manufacturer.''
::= { medicalDeviceControlEntry 3}

systemId OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ``This is the identifier that the manufacturer
gives to the medical device, like a serial number.''
::= { medicalDeviceControlEntry 4}

mdType OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
```

```
DESCRIPTION ‘‘This is the type of medical device between this
four: pulse oximeter, thermometer, weighing scale
or blood pressure monitor.’’
::= { medicalDeviceControlEntry 5}

protocolType OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This variable helps to know what kind of protocol
is being used to send the data from the device.’’
::= { medicalDeviceControlEntry 6}

configurationType OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This parameter gives the configuration of the
device. It could be standard of extended.’’
::= { medicalDeviceControlEntry 7}

updateRequestState OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is used by the manager to update the state
of one device connected to the CE. ’’
::= { medicalDeviceControlEntry 8}

updateRequestTechnicalData OBJECT-TYPE
SYNTAX Integer32
```

```
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is used by the manager to update the state
of one device connected to the CE. ’’
::= { medicalDeviceControlEntry 9}

deviceDate OBJECT-TYPE
SYNTAX DateTime
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘ This variable points when was the first time
that this device was connected to the CE.’’
::= { medicalDeviceControlEntry 10}



medicalDeviceDataTable OBJECT-TYPE
SYNTAX SEQUENCE OF MedicalDeviceDataEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘This table can store the data parameters of the
medical devices.’’
::= {medicalDeviceInfo 2}

medicalDeviceDataEntry OBJECT-TYPE
SYNTAX  MedicalDeviceDataEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the data parameters of each
medical device.’’
INDEX {idMDData, idCapture}
```

```
::= {medicalDeviceDataTable 1}

MedicalDeviceDataEntry ::= SEQUENCE {
idMDData  OCTET STRING,
idCapture OCTET STRING,
supplyType OCTET STRING,
batteryLevel Integer32,
remainderBattery Integer32,
errors OCTET STRING,
ownerMD OwnerString,
dateTimeData DateTime
}

idMDData OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the unique identifier of every medical
device connected to the compute engine. In this case determines
which is the device that made this sample. Represents the same
value that idMDControl.’’
::= { medicalDeviceDataEntry 1}

idCapture OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This object is useful to determine the order of
the captures of each medical device and have an
historic of the device.’’
::= { medicalDeviceDataEntry 2}
```

```
supplyType OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘The SupplyType shows if the device is using
batteries or is connected to the electrical net.’’
::= { medicalDeviceDataEntry 3}

batteryLevel OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This returns the percentage of battery that
left in the device.’’
::= { medicalDeviceDataEntry 4}

remainderBattery OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This returns the number of hours that the device
can work with the remaining battery.’’
::= { medicalDeviceDataEntry 5}

errors OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This parameter is activated when there is a error
at least in the device, and points to the id of the error in
```

```
the specificEerrorsTable.''
::= { medicalDeviceDataEntry 6}

ownerMD OBJECT-TYPE
SYNTAX OwnerString
ACCESS read-write
STATUS mandatory
DESCRIPTION ``This is the person that the measure belongs.''
::= { medicalDeviceDataEntry 7}

dateTimeData OBJECT-TYPE
SYNTAX DateTime
ACCESS read-only
STATUS mandatory
DESCRIPTION ``Here we save the time when the sample was taken.''
::= { medicalDeviceDataEntry 8}



medicalDeviceStateTable OBJECT-TYPE
SYNTAX SEQUENCE OF MedicalDeviceStateEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ``This table can store the change in the states of
the medical devices.''
::= {medicalDeviceInfo 3}

medicalDeviceStateEntry OBJECT-TYPE
SYNTAX  MedicalDeviceStateEntry
ACCESS not-accessible
STATUS mandatory
```

```
DESCRIPTION ‘‘Each entry saves the control parameters of each
medical device.’’
INDEX {idMDState,idState}
::= { medicalDeviceStateTable 1}

MedicalDeviceStateEntry ::= SEQUENCE {
idMDState OCTET STRING,
idState OCTET STRING,
deviceState DeviceStatus,
dateTimeState DateTime
}

idMDState OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the unique identifier of every medical
device connected to the compute engine. In this case determines
which is the device that made this sample. Represents the same
value that idMDControl.’’
::= { medicalDeviceStateEntry 1}

idState OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier od the new state of a
determinated MD.’’
::= { medicalDeviceStateEntry 2}

deviceState OBJECT-TYPE
```

```
SYNTAX DeviceStatus
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the state of the device, that can be
one of this: Disconnected (1), Connected (2),Unassociated (3),
Associating (4),Associated (5),Operating (6),Configuring (7) or
Disassociating(8).’’
::= { medicalDeviceStateEntry 3}

dateTimeState OBJECT-TYPE
SYNTAX DateTime
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Here we save the time when the state was changed.’’
::= { medicalDeviceStateEntry 8}



specificErrorsTable OBJECT-TYPE
SYNTAX SEQUENCE OF SpecificErrorsEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘This table can store errors of all type.’’
::= {medicalDeviceInfo 4}

specificErrorsEntry OBJECT-TYPE
SYNTAX  SpecificErrorsEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the control parameters of each
medical device.’’
```

```
INDEX {idError}
::= { specificErrorsTable 1}

SpecificErrorsEntry ::= SEQUENCE {
idMDError OCTET STRING,
idError   OCTET STRING,
deviceWorking OCTET STRING,
sensorWorking OCTET STRING,
sensorConnection OCTET STRING,
sensorJamming OCTET STRING,
signalFailures OwnerString
}

idMDError OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the unique identifier of every medical
device connected to the compute engine. In this case determines
which is the device that made this sample. Represents the same
value that idMDControl.’’
::= { specificErrorsEntry 1}

idError OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of each specific error.’’
::= { specificErrorsEntry 2}

deviceWorking OBJECT-TYPE
```

```
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Indicates if the device is working right or wrong.’’
::= { specificErrorsEntry 3}

sensorWorking OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Indicates if the sensor of the device is working
right or wrong.’’
::= { specificErrorsEntry 4}

sensorConnection OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Indicates if the sensor is well connected to the
device, and if it’s not, what is the problem: bad connection,
unsupported.’’
::= { specificErrorsEntry 5}

sensorJamming OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Indicates if there is some interferences between
the sensor and other signals, which cancel the measure we
have taken.’’
::= { specificErrorsEntry 6}
```

```
signalFailures OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘Indicates if the quality of the signal is good
enough or if it has some processing irregularities that cancel
the measure.’’
::= { specificErrorsEntry 7}



alarmTable OBJECT-TYPE
SYNTAX SEQUENCE OF AlarmEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘This table saves the alarms for the parameters we
want to have controlled.’’
::= {medicalDevicesManager 3}

alarmEntry OBJECT-TYPE
SYNTAX  AlarmEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the alarm for each parameter.’’
INDEX {idAlarm, idDevice}
::= { alarmTable 1}

AlarmEntry ::= SEQUENCE {
idDevice Integer32,
idAlarm OCTET STRING,
```

```
oidVariable OCTET STRING,
idEventAlarmValue OCTET STRING,
alarmValue OCTET STRING,
idEventUp Integer32,
idEventDown Integer32,
thresholdUp Integer32,
thresholdDown Integer32,
ownerAlarm OwnerString,
statusAlarm EntryStatus
}

idDevice OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This identifier point the device that belongs
the alarm.’’
::= { alarmEntry 1}

idAlarm OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of each alarm.’’
::= { alarmEntry 2}

oidVariable OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This indicates the OID of the variable that
```

```
is going to make the alarm activate, in case the thresholds
will be exceeded.''
::= { alarmEntry 3}

idEventAlarmValue OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ``It is the identifier of the event related to
the alarm activated by a determinate value of a parameter,
instead a threshold.''
::= { alarmEntry 4}

alarmValue OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ``This is the value that activates the alarm by
value.''
::= { alarmEntry 5}

idEventUp OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ``This is the identifier of the event that will
be activated if this alarm exceed the higher threshold.''
::= { alarmEntry 6}

idEventDown OBJECT-TYPE
SYNTAX Integer32
```

```
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of the event that will
be activated if this alarm exceed the lower threshold.’’
::= { alarmEntry 7}

thresholdUp OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the threshold that has to be always up
the signal. If not, the alarm is activated.’’
::= { alarmEntry 8}

thresholdDown OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the threshold that has to be always
down the signal. If not, the alarm is activated.’’
::= { alarmEntry 9}

ownerAlarm OBJECT-TYPE
SYNTAX OwnerString
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is responsible that has defined this alarm.’’
::= { alarmEntry 10}

statusAlarm OBJECT-TYPE
SYNTAX EntryStatus
```

```
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the status of the entry in the table.’’
::= { alarmEntry 11}



eventTables OBJECT IDENTIFIER ::= {medicalDevicesManager 4}

configurationEventTable OBJECT-TYPE
SYNTAX SEQUENCE OF ConfigurationEventEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘This object contains the tables of configuration
for the alarms we defined before.’’
::= {eventTables 1}

configurationEventEntry OBJECT-TYPE
SYNTAX  ConfigurationEventEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the event that are related to
the alarms.’’
INDEX {idEvent, idManager}
::= { configurationEventTable 1}

ConfigurationEventEntry ::= SEQUENCE {
idEvent Integer32,
eventType Integer32,
trapType PossibleTraps,
ownerEvent OCTET STRING,
```

```
statusEvent EntryStatus
}

idEvent OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of the event that will
be activated if this alarm exceed the higher threshold.’’
::= { configurationEventEntry 1}

eventType OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This indicates what happen when an event is
activated. We can send a trap, make an entry in the LogTable,
or both.’’
::= { configurationEventEntry 2}

trapType OBJECT-TYPE
SYNTAX PossibleTraps
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘Indicates what will be the message we send if
we decide to send a trap to the responsible.’’
::= { configurationEventEntry 3}

ownerEvent OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
```

```
STATUS mandatory
DESCRIPTION ‘‘This object saves the ID corresponding to the
ResponsibleTable that point the responsible to be warned.’’
::= { configurationEventEntry 4}

statusEvent OBJECT-TYPE
SYNTAX EntryStatus
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the status of the entry in the table.’’
::= { configurationEventEntry 5}



logTable OBJECT-TYPE
SYNTAX SEQUENCE OF LogEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘This object contains the tables of logs for
the events we decide to save a log.’’
::= {eventTables 2}


logEntry OBJECT-TYPE
SYNTAX  LogEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘Each entry saves the logs that are related
to the events.’’
INDEX {idEventLog, idLog}
::= { logTable 1}
```

```
LogEntry ::= SEQUENCE {
idEventLog Integer32,
idLog Integer32,
idAlarm Integer32,
logDescription OCTET STRING,
dateTimeLog DateTime
}

idEventLog OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of the event that
want to save a log.’’
::= { logEntry 1}

idLog OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of the current log,
to be able to identified it.’’
::= { logEntry 2}

idAlarm OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of the alarm that
activates the event that write this log.’’
```

```
::= { logEntry 3}

logDescription OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is a brief description of the cause
why the event has been activated.’’
::= { logEntry 4}

dateTimeLog OBJECT-TYPE
SYNTAX DateTime
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the moment when this log was recorded.’’
::= { logEntry 5}



managerTable OBJECT-TYPE
SYNTAX SEQUENCE OF ManagerEntry
ACCESS not-accessible
STATUS mandatory
DESCRIPTION ‘‘In this table we save the information related
to the managers of the net.’’
::= { medicalDevicesManager 5}

managerEntry OBJECT-TYPE
SYNTAX ManagerEntry
ACCESS not-accessible
STATUS mandatory
```

```
DESCRIPTION ‘‘This defines the entries of the table of
managers.’’
INDEX {idManager}
::={ managerTable 1}

ManagerEntry  ::= SEQUENCE {
idManager OCTET STRING,
owner OCTET STRING,
ip IpAddress,
phoneNumber Integer32,
email OCTET STRING,
workingplace OCTET STRING,
permission OCTET STRING,
nextFreeIndex Integer32,
statusManager EntryStatus
}

idManager OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is the identifier of this manager.’’
::= { managerEntry 1}

owner OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the name, with surnames, of this
manager.’’
::= { managerEntry 2}
```

```
ip OBJECT-TYPE
SYNTAX IpAddress
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the IP address we will use to contact
this resposible.’’
::= { managerEntry 3}

phoneNumber OBJECT-TYPE
SYNTAX Integer32
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the phone number of the manager.’’
::= { managerEntry 4}

email OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the e-mail address of the manager.’’
::= { managerEntry 5}

workingplace OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is an address we can use to mail
something to the manager.’’
::= { managerEntry 6}
```

```
permission OBJECT-TYPE
SYNTAX OCTET STRING
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This variable indicates the permission that
this manager have. If this value is ‘‘admin’’, he has full
permissions. If it’s ‘‘user’’ he can only read variable, not
set values.’’
::= { managerEntry 7}

nextFreeIndex OBJECT-TYPE
SYNTAX Integer32
ACCESS read-only
STATUS mandatory
DESCRIPTION ‘‘This is index after the ID of this manager, so
we can have an arranged table of managers.’’
::= { managerEntry 8}

statusManager OBJECT-TYPE
SYNTAX EntryStatus
ACCESS read-write
STATUS mandatory
DESCRIPTION ‘‘This is the status of the entry in the table.’’
::= {managerEntry 9}



medicalDevicesManagerTraps OBJECT IDENTIFIER
::={medicalDevicesManager 6}

systemOvercharged NOTIFICATION-TYPE
OBJECTS {bandwidthUse, CpuUse, hardDiskMemoryUse,
```

```
virtualMemoryUse, physicalMemoryUse, dateTimeControl}
STATUS current
DESCRIPTION ‘‘This trap is sent when some of the parameters of the
net that we are monitorising is too high, showing an overcharge in
the system.’’
::={medicalDevicesManagerTraps 1}

wrongDeviceWorking NOTIFICATION-TYPE
OBJECTS {idMDData, idCapture, DateTimeData}
STATUS current
DESCRIPTION ‘‘This trap warns that the indicated measure is out of
the limits stablished, and that probably means a bad working
of the device.’’
::={medicalDevicesManagerTraps 2}

specificError NOTIFICATION-TYPE
OBJECTS {idMDError, idError, deviceWorking, sensorWorking,
deviceConnection, sensorJamming, signalFailures}
STATUS current
DESCRIPTION ‘‘This trap is useful when a specific Error of a device
occurs.’’
::={medicalDevicesManagerTraps 3}

newMD NOTIFICATION-TYPE
OBJECTS {idMDControl, manufacturer, model, systemId, mdType,
protocolType, deviceDate}
STATUS current
DESCRIPTION ‘‘This trap is useful when a specific Error of a device
occurs.’’
::={medicalDevicesManagerTraps 4}
```

```
warning NOTIFICATION-TYPE
OBJECTS {}
STATUS current
DESCRIPTION ‘‘This trap is designed for the alarmas that are not
covered by the others traps.’’
::= {medicalDevicesManagerTraps 5}

END
```

Anexo F

Diagrama de Gantt

A continuación se presentan las tareas que aparecen en el diagrama de Gantt del presente Proyecto Fin de Carrera mostrado en la figura F.1:

1. Documentación de arquitectura SNMP.
2. Documentación sobre el estado del arte.
3. Implementación del bloque de comunicaciones SNMP.
4. Test del funcionamiento del bloque de comunicaciones SNMP.
5. Definición formal de la MIB.
6. Implementación física de la MIB.
7. Test de funcionamiento de la MIB.
8. Revisión de las especificaciones del estándar X.73.
9. Implementación del proxy agente-manager X.73.
10. Test del funcionamiento del proxy agente-manager X.73.
11. Integración de los bloques.
12. Implementación del interfaz de simulación del manager X.73.
13. Implementación del interfaz del agente.

14. Test de funcionamiento de los interfaces.

15. Redacción de la memoria.

16. Ejecución de las pruebas y análisis de los resultados.

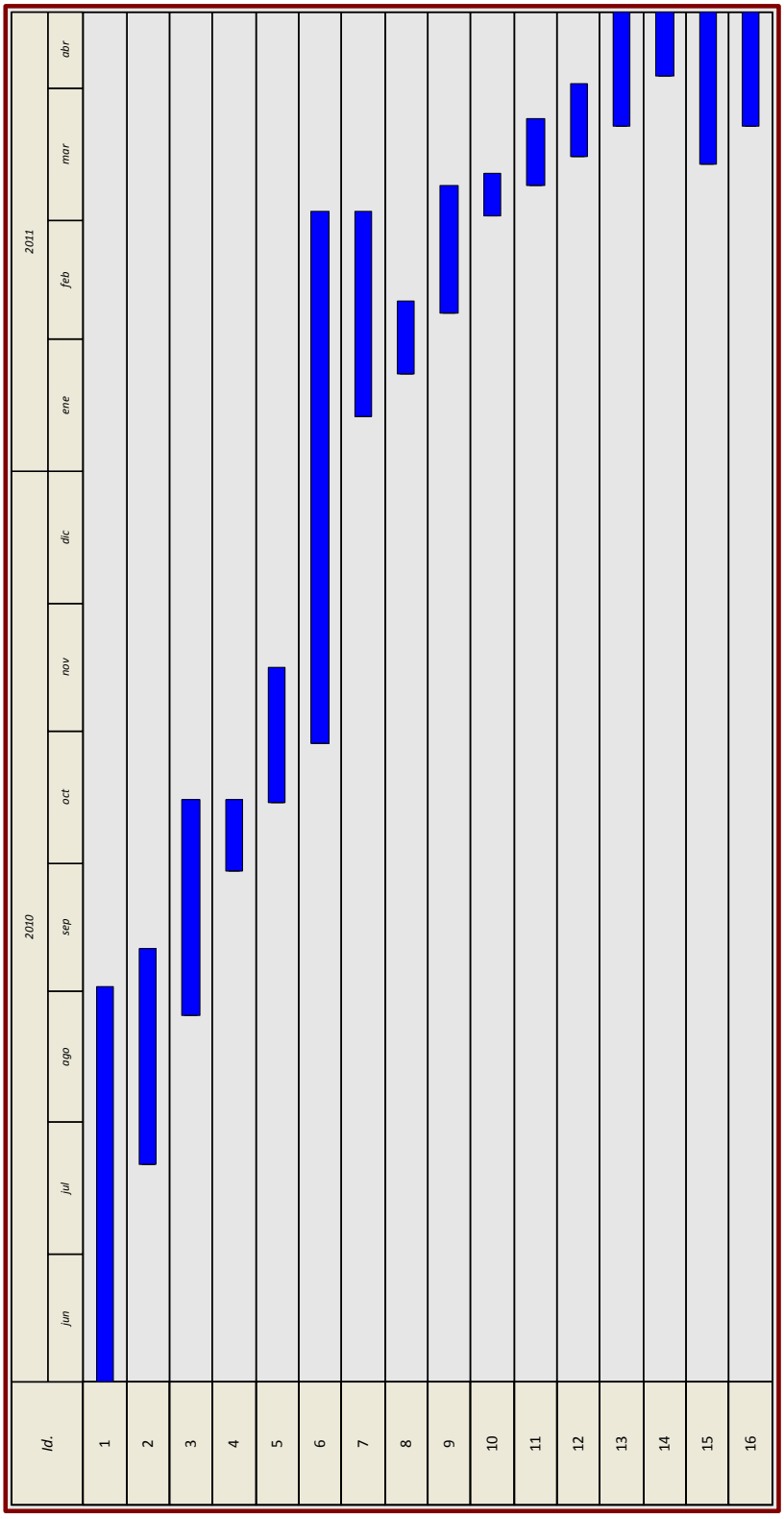

Figura F.1. Diagrama de Gantt del Proyecto Fin de Carrera.