Universidad de Valladolid

Trabajo de Fin de Grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación

Mención en Telemática

HERRAMIENTA GRÁFICA DE GESTIÓN SNMP Y MANEJO DE MIB

AUTOR: Gonzalo Lezcano Hermoso
TUTOR: Federico J. Simmross Wattenberg

Mayo, 2018

RESUMEN DEL PROYECTO

En los albores de la interconexión de redes, aspectos como la monitorización o la gestión de éstas no habían sido siquiera planteados. La falta de un protocolo consistente en este sentido y la creciente heterogeneidad de las redes motivaron la aparición de SNMP (*Simple Network Management Protocol*) en el año 1988. Realizando una gestión de la información contenida en MIB (*Management Information Base*) y favorecido por el acuñamiento de TCP/IP, SNMP se ha convertido en uno de los protocolos más longevos y utilizados de Internet. Ante esta hegemonía, muchos fabricantes se decidieron a introducir su propia información propietaria en MIB, construyendo aplicaciones a partir de ello.

Sin embargo, la inexistencia de soluciones de código libre en este sentido es palpable y supone un problema a resolver para el administrador de red de una empresa que decida no invertir un gran volumen económico en un sistema de gestión propietario basado en SNMP.

Este Trabajo de Fin de Grado responde a la necesidad que acabamos de exponer, constituyendo una herramienta gráfica de gestión SNMP y manejo de MIB.

ABSTRACT

At the dawn of networking, issues such as the monitoring or management functions had not even been considered. The lack of a protocol and the growing network heterogeneity were the reason for the appearence of SNMP (Simple Network Management Protocol) in 1988. By doing an information management included on MIB (Management Information Base), it has becomed one of the most long-lived and used protocols on the Internet due to the victory of TCP/IP instead of OSI. As a consequence of this hegemony, lots of manufacturers decided to introduce their own proprietary information in MIB, building applications from it.

Nevertheless, the absence of open source solutions is obvious. Furthermore, it poses a great problem to be solved by the system administrator of the companies which decide not to do big economic investments in these solutions.

This End of Degree Paper intends to be an answer for this necessity, by developing a graphic tool whose purpose is to manage a network via SNMP and MIB.

PALABRAS CLAVE

Gestión de red, SNMP, agente, NMS, MIB, GUI.

Para ti

Agradecimientos

A Olga, por no dejar que me relajase en los momentos en que más olvidado tenía este proyecto.

A José Antonio («Tony») Grijalva, por creer siempre en mí, y por perder su tiempo en intentar explicarme todo cuanto necesitaba.

A mis compañeros de Nokia, por hacerme ver con perspectiva lo que una herramienta de gestión de red basada en SNMP debía realizar.

A Miriam, por aguantarme durante toda la carrera. Por todas esas mañanas, tardes y noches perdiendo nuestra vida en sacar esta titulación. Por los ratos en el aulario, biblioteca de ciencias, la antigua de nuestra escuela, nuestra sala, los laboratorios o incluso en la casa de *Mordor*. Al final acabamos esta locura.

A mi familia, por todos los esfuerzos que han tenido que soportar para que obtuviera la titulación, incluidos estos últimos dos meses de«lastre» continuo.

Y, por supuesto, a mi tutor. Creo que hemos construido un programa de calidad y con un futuro que debemos recorrer.

Índice general

Índice de figuras **XIII**

Índice de tablas **XVII**

1. Introducción **1**
1.1. Motivación 1
1.2. Objetivos 3
1.3. Metodología y fases del proyecto 4
1.4. Medios 5
1.4.1. *Hardware* 5
1.4.2. *Software* 5
1.5. Estructura de la memoria 6

2. El protocolo SNMP **9**
2.1. Historia del protocolo 10
2.1.1. Antecedentes 10
2.1.2. 1988: El surgimiento de SNMP 11
2.1.3. Evolución y distintas versiones 13
2.1.4. Uso actual 15
2.2. Características 16
2.2.1. Introducción y características generales 16
2.2.2. Manejo de la información: SMI, MIB y OID 17
2.2.3. Envíos del agente: *Traps* 21
2.2.4. Envíos del servidor: *set, get, get-next* 24
2.2.5. Autenticación: *Communities, user-based, v3* 29
2.3. Net-SNMP 34

3. Tecnologías auxiliares **37**
3.1. Lenguaje de programación 37
3.1.1. Librerías SNMP en los lenguajes objeto de análisis . . 39
3.1.2. Librerías gráficas en los lenguajes objeto de análisis . . 40

3.1.3. Discusión 41
3.2. *PyQt* como herramienta gráfica 42
3.3. *SQLite* para nuestra base de datos 42
3.4. Otras tecnologías 43

4. Análisis y diseño de la aplicación 45
4.1. Introducción 45
4.2. Análisis 46
4.2.1. Análisis de requisitos 46
4.2.2. Análisis de casos de uso 49
4.3. Diseño 56
4.3.1. Diseño de clases 57
4.3.2. Diseño de la base de datos 62
4.3.3. Diseño del fichero de configuración 63
4.3.4. Diseño de la interfaz gráfica de usuario 64

5. Desarrollo e implementación 69
5.1. Introducción 69
5.2. Desarrollo de los distintos casos de uso 69
5.2.1. Desarrollo del caso de uso CambiarValores 69
5.2.2. Desarrollo del caso de uso EnviarAlarma 74
5.2.3. Desarrollo del caso de uso BuscarElemento 76
5.2.4. Desarrollo del caso de uso ResincronizarTablaElementos 83
5.2.5. Desarrollo del caso de uso AplicarConfiguración 89
5.2.6. Desarrollo del caso de uso GenerarInforme 90
5.2.7. Desarrollo del caso de uso ModificarConfiguración . . . 93
5.2.8. Desarrollo del caso de uso DecodificarOID 96
5.2.9. Desarrollo del caso de uso CargarMIB 98

6. Pruebas 99
6.1. Introducción 99
6.2. Cumplimiento de requisitos 100
6.2.1. Requisitos funcionales 100
6.2.2. Requisitos no funcionales 110

7. Conclusiones y líneas futuras 117
7.1. Conclusiones 117
7.2. Líneas futuras 119

A. *dialogosAdicionales.py* 121

B. *busquedas.py* 129

C. *controlador.py* **137**

D. *ventanaPrincipal.pyw* **145**

E. *editarConf.py* **153**

F. *snmpserver.py* **159**

Índice de figuras

2.1. Arquitectura y comunicaciones de SNMP 18
2.2. Ejemplo de árbol SMI 19
2.3. *coldStart v1-trap* 23
2.4. *coldStart v2-trap* 23
2.5. *set-request* 24
2.6. *set-response* 25
2.7. *get-request* 27
2.8. *get-response* 27
2.9. *Walk* sobre el árbol MIB 28
2.10. *get-next* sobre *SNMPv2-MIB::system* 29
2.11. Procesamiento de mensajes con USM 32
2.12. Toma de decisiones VACM 33
2.13. *snmpwalk* sobre *SNMPv2-MIB::system* 35

3.1. Comparativa del uso de lenguajes de programación en el periodo 2004-2018 40

4.1. Diagrama de casos de uso 56
4.2. Estructura de ficheros de la aplicación 59
4.3. Diagrama de clases 61
4.4. Diseño de la base de datos 62
4.5. Ejemplo de fichero de configuración 64
4.6. Boceto de la interfaz gráfica 67

5.1. Desarrollo del caso de uso CambiarValores *acceptSet* 70
5.2. Desarrollo del caso de uso CambiarValores *controlCambio* . . . 71
5.3. Desarrollo del caso de uso CambiarValores. Llamada al controlador central (1) 71
5.4. Desarrollo del caso de uso CambiarValores. Llamada al controlador central (2) 72
5.5. Desarrollo del caso de uso CambiarValores *determinarComm* . 73

5.6. Desarrollo del caso de uso CambiarValores. Comprobación de *community* 73
5.7. Desarrollo del caso de uso CambiarValores. Llamada al servidor 73
5.8. Desarrollo del caso de uso CambiarValores. *snmpset* 74
5.9. Desarrollo del caso de uso EnviarAlarma. Creación de un nuevo proceso 75
5.10. Desarrollo del caso de uso EnviarAlarma. Apertura de puerto . 75
5.11. Desarrollo del caso de uso EnviarAlarma. Recepción de *traps* . 76
5.12. Desarrollo del caso de uso EnviarAlarma. Representación . . . 77
5.13. Desarrollo del caso de uso BuscarElemento. Diálogo inicial comprimido 78
5.14. Desarrollo del caso de uso BuscarElemento. Diálogo inicial expandido 79
5.15. Desarrollo del caso de uso BuscarElemento. Campos inicialmente ocultos 79
5.16. Desarrollo del caso de uso BuscarElemento. Método *addField* . 80
5.17. Desarrollo del caso de uso BuscarElemento. Método *deleteField* 81
5.18. Desarrollo del caso de uso BuscarElemento. Traducción de valores lógicos 81
5.19. Desarrollo del caso de uso BuscarElemento. Método *buscarValores* 82
5.20. Desarrollo del caso de uso BuscarElemento. Ejecución de la consulta 83
5.21. Desarrollo del caso de uso BuscarElemento. Número de filas . 83
5.22. Desarrollo del caso de uso BuscarElemento. Representación del resultado 84
5.23. Desarrollo del caso de uso ResincronizarTablaElementos. Lectura del fichero de configuración 85
5.24. Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (1) 86
5.25. Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (2) 86
5.26. Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (3) 86
5.27. Desarrollo del caso de uso ResincronizarTablaElementos. Algoritmo de descubrimiento 87
5.28. Desarrollo del caso de uso ResincronizarTablaElementos. Método *snmpget* 87
5.29. Desarrollo del caso de uso ResincronizarTablaElementos. Actualización de la base de datos 88

5.30. Desarrollo del caso de uso ResincronizarTablaElementos. Método *wakeUp* 89
5.31. Desarrollo del caso de uso AplicarConfiguración 90
5.32. Desarrollo del caso de uso GenerarInforme. Manejo de peticiones 91
5.33. Desarrollo del caso de uso GenerarInforme. Método *peticiones* 91
5.34. Desarrollo del caso de uso GenerarInforme. Método *snmpwalk* 93
5.35. Desarrollo del caso de uso GenerarInforme. Algoritmo para representar el árbol. Primera fila 94
5.36. Desarrollo del caso de uso GenerarInforme. Algoritmo para representar el árbol. Filas distintas a la primera 94
5.37. Desarrollo del caso de uso ModificarConfiguración. Apariencia del diálogo 95
5.38. Desarrollo del caso de uso ModificarConfiguración. Módulos MIB 95
5.39. Desarrollo del caso de uso ModificarConfiguración. Actualización del fichero 96
5.40. Desarrollo del caso de uso DecodificarOID. *getLabel* 97
5.41. Desarrollo del caso de uso DecodificarOID. Método *decodificarOID* 97
5.42. Desarrollo del caso de uso CargarMIB 98

6.1. Cumplimiento de FRQ-001. Selección de elemento 101
6.2. Cumplimiento de FRQ-001. Cambiar parámetros 101
6.3. Cumplimiento de FRQ-001. *sysName* original 102
6.4. Cumplimiento de FRQ-001. Introduciendo el cambio 102
6.5. Cumplimiento de FRQ-001. Éxito en el cambio 102
6.6. Cumplimiento de FRQ-001. Valor cambiado 103
6.7. Cumplimiento de FRQ-002. Ninguna notificación 104
6.8. Cumplimiento de FRQ-002. Reinicio del agente 104
6.9. Cumplimiento de FRQ-002. Alarmas representadas 105
6.10. Cumplimiento de FRQ-003. Icono de búsqueda 106
6.11. Cumplimiento de FRQ-003. Parámetros de búsqueda 107
6.12. Cumplimiento de FRQ-003. Resultado de la búsqueda 107
6.13. Cumplimiento de FRQ-004. Petición 107
6.14. Cumplimiento de FRQ-004. Resultados 108
6.15. Cumplimiento de FRQ-005. *config.xml* 109
6.16. Cumplimiento de FRQ-005. Resultado tras carga inicial 109
6.17. Cumplimiento de NFRQ-003. Decodificación en la ventana principal 111
6.18. Cumplimiento de NFRQ-004. Opción «Editar configuración» . 112
6.19. Cumplimiento de NFRQ-004. Interfaz de edición 113

6.20. Cumplimiento de NFRQ-004. Configuración modificada 113
6.21. Cumplimiento de NFRQ-004. Cambios mostrados en *config.xml* 114
6.22. Cumplimiento de NFRQ-004. Cambios mostrados en la interfaz gráfica 114
6.23. Cumplimiento de NFRQ-005. *Walk* completo 115
6.24. Cumplimiento de NFRQ-005. Resultados compactos 115

Índice de tablas

2.1. Tipos de datos SMI 21
2.2. *Traps* genéricas 22
2.3. Códigos de error SNMP 26

4.1. Requisitos funcionales. 47
4.2. Requisitos no funcionales. 48
4.3. Caso de uso CambiarValores. 49
4.4. Caso de uso EnviarAlarma. 50
4.5. Caso de uso BuscarElemento. 51
4.6. Caso de uso ResincronizarTablaElementos. 52
4.7. Caso de uso AplicarConfiguración. 53
4.8. Caso de uso GenerarInforme. 54
4.9. Caso de uso ModificarConfiguración. 54
4.10. Caso de uso DecodificarOID. 55
4.11. Caso de uso CargarMIB. 55

Capítulo 1

Introducción

Este capítulo, como su propio nombre indica, constituye la introducción del documento. Para realizarla adecuadamente, dividiremos este apartado en cinco secciones en las que iremos desarrollando la motivación del trabajo, los objetivos que se pretenden cumplir, la metodología empleada, los medios utilizados y la estructura de la memoria.

1.1. Motivación

En los primeros años de la interconexión de redes, cuando éstas eran básicamente elementos de investigación, términos como los de gestión o monitorización de red eran desconocidos y no habían sido siquiera planteados. Debido a ello, determinar la causa de un comportamiento inesperado en la red era una tarea ardua, teniendo que involucrar a múltiples actores en limitadas labores más allá de realizar una serie de *pings* o desplazarse físicamente a los terminales afectados a fin de comprobar su estado [1]. Así, podemos ver un ejemplo de toda la investigación *post-mortem* que había de ser llevada a cabo a principios de los 80 ante un *outage* en ARPANET (*Advanced Research Projects Agency NETwork*) en [2].

La dificultad para predecir estos comportamientos inesperados, determinar sus causas y la simple gestión de los elementos de la red motivaba la aparición de algún mecanismo que ayudase en esta dirección. Además, dada la creciente heterogeneidad de las redes, con la aparición de múltiples fabri-

cantes y diversas topologías, se apuntaba a la necesidad de una solución no propietaria para la gestión de red. Estos hechos motivan la aparición del protocolo SNMP (*Simple Network Management Protocol*) en el año 1988 [3, 4].

SNMP emplea, para el manejo de la información la estructura SMI (*Structure of Management Information*) y las MIB (*Management Information Base*), cuya arquitectura jerárquica hace que la representación en forma de árbol sea una manera muy útil de gestionar y visualizar la información. También estas estucturas son maduras, pues SMIv2 y MIB-II, aún utilizadas con modificaciones menores, fueron planteadas en los años 1993 y 1991 respectivamente [5, 6].

Este protocolo se convirtió rápidamente en indispensable para la gestión de red, tanto por la necesidad previa existente como por la simplicidad que de él se desprende, hecho que ha favorecido su uso aún hoy en día, pese a ser un protocolo maduro del que su última versión fue aceptada como estándar en 2002 [3]. Las distintas empresas del sector comenzaron a introducir agentes SNMP en sus productos de forma masiva a comienzos de los años 90 [7], y aún hoy continúan presentes, formando parte de los productos de nuevo desarrollo [8–10].

Así, fueron surgiendo múltiples aplicaciones para la gestión de red basadas en SNMP, aunque en su mayor parte fueron propietarias y con un coste usualmente demasiado elevado para redes de dimensiones reducidas o con un presupuesto humilde. Proveedores como Nokia o Cisco ofrecen potentes soluciones para la gestión de red apoyándose en SNMP, como es el caso de NFM-P (*Network Functions Manager for Packet*) en Nokia [11] o *Unified Communications Manager Managed Services* para Cisco [12].

Sin embargo, en el caso de las soluciones de código abierto apreciamos en ocasiones falta de ambición, como el caso de *tkmib*, cuya falta de documentación y funcionalidades [13] hace que su uso no sea recomendable, pudiendo preferir en múltiples ocasiones la gestión de uno o varios agentes vía CLI (*Command Line Interface*). Además, el escaso abanico de elección en este ámbito para soluciones libres y/o gratuitas que realicen esta tarea de manera íntegra resulta flagrante. Por ejemplo, podríamos encontrarnos ante soluciones como MRTG (*Multi Router Traffic Grapher*), que emplea el protocolo SNMP y es utilizada de manera intensiva, pero únicamente nos serviría para realizar gráficos sobre la carga de una interfaz, sin tener en cuenta el resto de posibilidades que ofrece nuestro protocolo [14].

En virtud de lo expuesto, este Trabajo de Fin de Grado pretende suplir estas deficiencias para, sin llegar a las funcionalidades de las aplicaciones propietarias mencionadas, proporcionar una herramienta gráfica útil, intuitiva y bien documentada para la gestión SNMP y el manejo de MIB (*Management Information Base*), pretendiendo hacer más fácil el día a día del administrador de red que no pueda disponer de una de las soluciones propietarias de los principales proveedores.

1.2. Objetivos

El objetivo fundamental del presente Trabajo de Fin de Grado es la programación de una herramienta gráfica de gestión SNMP y manejo de MIB funcional y útil para el día a día en la administración de red. Para ello, se deben satisfacer una serie de requisitos tales como:

- **Representar el árbol MIB.** El programa debe permitir dibujar el árbol MIB para cada agente, de una forma agradable e intuitiva para el usuario.
- **Identificar los nodos hoja.** Dentro de la representación del árbol MIB, la interfaz gráfica debe presentar al usuario una diferenciación entre nodos rama (contenedores de nodos) y nodos hoja (contenedores de información).
- **Realizar búsquedas.** El programa debe permitir realizar búsquedas entre los elementos gestionados.
- **Descubrir dispositivos.** La herramienta ha de proveer un método de descubrimiento de dispositivos gestionables en la red.
- **Leer y escribir un OID en concreto.** El programa debe facilitar la lectura y escritura de uno o varios OIDs a través de la interfaz gráfica.
- **Leer un subárbol al completo.** La herramienta debe permitir leer un módulo MIB al completo, representándolo en forma de árbol.
- **Recibir «trampas» SNMP.** El programa debe recibir, procesar y mostrar al usuario las 'trampas' SNMP enviadas por los agentes de los elementos gestionados.

- **Configurar parámetros de funcionamiento.** La herramienta debe permitir al usuario alterar ciertos parámetros de configuración a través de la interfaz gráfica.

Todos estos objetivos serán analizados en el capítulo cuarto de la presente memoria, seleccionando una serie de requisitos funcionales y no funcionales que deberán ser satisfechos a fin de cumplir los objetivos que acabamos de enunciar.

1.3. Metodología y fases del proyecto

Para la consecución de los objetivos enunciados en el apartado previo se ha seguido una metodología de trabajo, que puede ser dividida en las siguientes fases:

1. **Documentación acerca del protocolo SNMP.** A fin de obtener una mayor profundidad de conocimientos de la que previamente se disponía y entendiendo como parte indispensable del trabajo fin de grado la correcta asimilación de las distintas singularidades del protocolo, se realizará una primera parte de estudio del protocolo SNMP, tecnología base e hilo conductor de todo el trabajo.

2. **Análisis de objetivos.** Una vez interiorizada la naturaleza del protocolo, se realizará un análisis de los objetivos a cumplir, los cuales han sido expuestos en la sección anterior.

3. **Elección de las tecnologías auxiliares a emplear.** Vistos los objetivos, se realizará un análisis, en función de ellos, de las tecnologías auxiliares a emplear para lograrlos.

4. **Formación en tecnologías auxiliares.** Para realizar un correcto desarrollo del proyecto, se considera necesario un periodo de formación en las tecnologías auxiliares necesarias para la programación de la herramienta.

5. **Ingeniería de *software*.** Se llevará a cabo un proceso de ingeniería de *software*, incluyendo, como parte del análisis, los análisis de requisitos y de casos de uso e incluyendo, como parte del diseño, el diagrama de clases.

6. **Desarrollo de la aplicación.** Se trata de la escritura y desarrollo de la aplicación en función del proceso de ingeniería de *software* llevado a cabo previamente.

7. **Fase de pruebas.** Tras la escritura de la herramienta, se llevan a cabo una serie de pruebas, destinadas a corregir los posibles errores que pudieran existir y a mejorar el comportamiento de la aplicación.

8. **Desarrollo y escritura de la memoria.** En esta última fase del proyecto, transversal, pues distintas partes de la presente memoria han sido escritas durante las fases previas, se ha documentado todo el trabajo anterior, dotando al trabajo de la apariencia que aquí se presenta.

1.4. Medios

Los medios empleados para satisfacer los objetivos enunciados en la sección 1.2 y en virtud a la metodología indicada en la sección 1.3, podemos definirlos entre *hardware* y *software* y son los siguientes:

1.4.1. *Hardware*

- Ordenador portátil *msi® PE60 6QE*:
  - Procesador *Intel® Core™ i7-6700HQ (2.6 GHz, 3MB).*
  - Memoria RAM 16GB DDR4 SODIMM (2x8GB).
  - Disco duro mecánico 1 TB (7200 rpm S-ATA) + 128 GB SSD.
  - Controlador gráfico *Nvidia® GeForce GTX 960M 2GB GDDR5.*
- Impresora *Brother® MFC-L2700DW*, gestionable por SNMP.

1.4.2. *Software*

- Sistema operativo del ordenador portátil: *Ubuntu 16.04.3 LTS.*
- Hipervisor de tipo 2 *Oracle® VM VirtualBox 5.2.0_RC1 r118201 (Qt5.6.1).*
- Máquina virtual alojada en nuestro hipervisor con sistema operativo *CentOS 7.*

- Agente SNMP *snmpd* instalado en la máquina virtual que acaba de ser nombrada.

1.5. Estructura de la memoria

A continuación se enuncian los diferentes capítulos de que consta la memoria, siendo brevemente explicados. Se disponen de tal forma que podemos dividirlos en bloques. De esta manera, los primeros tres capítulos (incluyendo el presente) se dedican a realizar una introducción del proyecto, detallando el estado del arte y dando un marco para la ingeniería de *software* y programación posterior. Los siguientes dos capítulos se dedican precisamente a la ingeniería de software y programación previamente mencionadas, mientras que en el sexto se presentan los resultados mediante una serie de pruebas y el último capítulo versa acerca de las conclusiones y líneas futuras que se desprenden del trabajo. Además, se presentarán una serie de apéndices dispuestos a complementar ciertas partes de la memoria con porciones del código de la herramienta.

Así, entre los capítulos destinados a analizar el estado del arte tenemos:

- **Capítulo segundo:** *el protocolo SNMP.* Se dedica a realizar una introducción histórica del protocolo base de nuestro trabajo, analizando brevemente sus características y utilidades para que el lector se encuentre familiarizado con su uso y sus implicaciones a lo largo de la presente memoria.
- **Capítulo tercero:** *tecnologías auxiliares.* En este capítulo se analizarán las diferentes tecnologías que nos acompañarán en nuestro trabajo. Así, se discutirá especialmente el lenguaje de programación a utilizar, las bibliotecas de la interfaz gráfica de usuario y el sistema de gestión de bases de datos.

Entrando ya en el análisis y desarrollo de la aplicación encontramos los dos siguientes capítulos:

- **Capítulo cuarto:** *Análisis y diseño de la aplicación.* Este capítulo mostrará los aspectos relacionados con la ingeniería de *software* realizada para la consecución de los objetivos descritos en la sección 1.2.

Así, se mostrará el análisis de requisitos, divididos entre funcionales y no funcionales; el análisis de casos de uso, con un diagrama relacionando éstos con los diferentes actores; y, por último, un diagrama de clases.

- **Capítulo quinto:** *Desarrollo e implementación.* Partiendo del análisis realizado en el capítulo que lo precede, se mostrarán los aspectos más relevantes del desarrollo e implementación de la herramienta.

Los últimos dos capítulos se dedican a las pruebas realizadas para comprobar el correcto funcionamiento de la aplicación y a la extracción de conclusiones:

- **Capítulo sexto:** *Pruebas.* En este capítulo se evalúa el cumplimiento de los objetivos iniciales y se muestran los resultados y pruebas más relevantes a que ha sido sometida la aplicación.
- **Capítulo séptimo:** *Conclusiones y líneas futuras.* En el capítulo final se muestran las conclusiones obtenidas tras realizar todo el trabajo fin de grado. Así mismo, se extraen unas líneas futuras por las que el proyecto podría avanzar.

En cuanto a los apéndices, nos encontramos con diferentes ficheros íntegros con código de la aplicación, incluidos aquí a fin de complementar el capítulo 5 en ciertos aspectos en los que el lector podría querer disponer de información adicional más allá de las capturas expuestas. De este modo, nos encontramos ante los siguientes apéndices:

- **Apéndice A:** *dialogosAdicionales.py.*
- **Apéndice B:** *busquedas.py*
- **Apéndice C:** *controlador.py*
- **Apéndice D:** *ventanaPrincipal.pyw*
- **Apéndice E:** *editarConf.py*
- **Apéndice F:** *snmpserver.py*

Capítulo 2

El protocolo SNMP

A lo largo del presente capítulo se pretende dar al lector una imagen general, algo más profunda de lo esbozado en la introducción, del protocolo SNMP, hilo principal y motivador del trabajo. Así, se expondrá una perspectiva histórica, en la que se podrá interiorizar la necesidad previa a la aparición de SNMP, las razones que han llevado a la aparición de nuevas versiones, la preocupación por dotar a este protocolo de los mecanismos de seguridad adecuados y el uso que de él se hace actualmente. Posteriormente, tendremos una sección en la que «diseccionaremos» el protocolo, analizando sus características y funcionalidades principales. Explicaremos cómo maneja la información, así como las principales operaciones llevadas a cabo por este protocolo. También repasaremos las distintas opciones de autenticación que se ponen a disposición con las diferentes evoluciones del protocolo. Por último en este capítulo, trataremos el proyecto Net-SNMP, que ofrece la implementación más empleada del protocolo SNMP para el mundo UNIX [7].

Con todo esto se pretende que, al finalizar el capítulo, el lector o la lectora se encuentre completamente inmerso en las operaciones a realizar por nuestra aplicación, así como que comprenda el porqué de su necesidad, sirviendo de justificación para las funcionalidades de nuestra herramienta gráfica, mostradas en los capítulos 4 y posteriores.

2.1. Historia del protocolo

En la sección que comienza, vamos a estudiar los orígenes y evolución de SNMP, comprendiendo su motivación inicial y avanzando hasta analizar su uso actual.

2.1.1. Antecedentes

El origen del protocolo SNMP se remonta a los problemas surgidos en los inicios de Internet. En el marco del desarrollo de ARPANET se tiene en cuenta ya la existencia de múltiples y diferentes redes de conmutación de paquetes que debían ser comunicadas [15]. Esta tarea se pretendía solucionar con lo que en su momento llamaron «pasarelas» o *gateways*, dando paso a lo que hoy en día conocemos como *routers*. Así, la resultante red «concatenada» (más tarde llamada «interconectada») sería mucho más potente y útil, dado que se abría paso una amplia variedad de servicios, restringida en el caso de las soluciones ofrecidas dentro de cada propia red [16].

De este modo, entre estos servicios podemos contar también con ciertos mecanismos para la gestión de red, como por ejemplo el relativo a la gestión de redes LAN (*Local Area Network*) de IBM, que puede verse en [17].

Además, como se mencionó en la sección 1.1, la investigación o prevención de los problemas en los orígenes del *networking* era una labor harto complicada, que señalaba claramente la necesidad de mecanismos de prevención o investigación que hicieran de la gestión de red una opción real. La simple disposición de ICMP (*Internet Control Message Protocol*) no resultaba suficiente en los problemas no evidentes y el crecimiento que empezaba ya a darse a finales de los 80 no hacía sino agravar los problemas derivados de basar la gestión y la monitorización de red en un protocolo con severas limitaciones [18].

Como hemos visto, comenzaban a surgir soluciones propietarias para sus propias redes pero, dado el inicio de la interconexión de redes que finalmente acabaría con el nacimiento de Internet, se precisaba de un protocolo que resolviese esta heterogeneidad, pudiendo ser empleado para múltiples fabricantes y topología. Es en este contexto donde, en **noviembre de 1987**, nace **SGMP** (*Simple Gateway Monitoring Protocol*) [19].

Este protocolo nace justamente con unos objetivos de diseño destinados a paliar los retos evidentes descritos. Los autores del RFC (*Request For Comments*) que lo especifica [20] dividen estos objetivos en 8, con un deseo claro, el de minimizar la complejidad y número de funciones de gestión que una *gateway* debía realizar por su cuenta para solventar los problemas derivados de la heterogeneidad de la red.

De este modo, todos sus objetivos van en la línea de conseguir una solución simple, escalable e independiente de la arquitectura y mecanismos propietarios.

Para lograr su cometido, definen una serie de mensajes para el protocolo, a saber:

- *Get Request.*
- *Trap Request.*
- *Set Request.*

Además, esta propuesta de finales de los 80 para la gestión de red, es entendida por sus autores como «únicamente una respuesta temporal para las inmediatas necesidades de monitorización de las *gateways*» [20]. De acuerdo a esta motivación, y tomando como base SGMP, 1988 ve nacer al que será uno de los protocolos más longevos de Internet: SNMP.

2.1.2. 1988: El surgimiento de SNMP

Como predijeron Davin y sus colegas con el lanzamiento de SGMP, este protocolo se quedó corto pronto. El incremento exponencial del número de equipos conectados a Internet a finales de los 80 y comienzos de los 90 hacía que al prototipo inicial desarrollado por estos investigadores le fuese necesaria una evolución.

Así, en agosto de 1988, surge **SNMP (*Simple Network Management Protocol*)**, como una versión mejorada de SGMP desarrollada por el mismo equipo [21].

Como cabría esperar, fueron presentadas más propuestas. Por lo tanto, es razonable preguntarse cuál fue la razón de que SNMP triunfase y perdurara en el tiempo hasta nuestros días, 30 años después.

De entre las diferentes alternativas presentadas, William Stallings cree oportuno destacar tres en [18]. Estas son:

- *High-Level Entity-Management System (HEMS).* Generalización del protocolo HMP (*Host Monitoring Protocol*), uno de los primeros protocolos de gestión de red usados en Internet [4].
- *Simple Network Management Protocol (SNMP).*
- *CMIP over TCP/IP (CMOT).* Intento de incorporar, de la manera más extensa posible, el protocolo CMIP (*Common Management Information Protocol*), los servicios y la estructura de base de datos que habían sido estandarizados por la ISO (*International Organization for Standarization*) para la gestión de red.

El razonamiento inicial de la IAB (*Internet Activities Board*) fue que SNMP debía ser una solución a desarrollar a corto plazo y con un recorrido más bien limitado, mientras se producía la transición «necesaria» de TCP/IP al modelo OSI (*Open System Interconnection*). Las acuciantes necesidades de un protocolo de estas características para la red TCP/IP inclinaron la balanza a favor de SNMP. Este protocolo podría ser rápidamente desarrollado, ofreciendo algunas herramientas básicas de gestión, mientras que soportaba el avance del ámbito de la gestión de red. Así, aunque sobre el papel HEMS presentaba mejores características que nuestro protocolo, SNMP fue rápidamente desarrollado y, vista la fracasada transición a OSI, SNMP perduró en el tiempo, siendo sus deficiencias iniciales subsanadas progresivamente con la aparición de distintas versiones y actualizaciones del protocolo, como veremos en la siguiente subsección.

Además, si se conseguía que CMIP (pensado para ser implementado con OSI) pudiese funcionar sobre TCP/IP, CMOT podría ser empleado incluso antes de la transición a OSI.

Para solidificar esta estrategia, la IAB dictaminó que tanto SNMP como CMOT debían emplear el mismo sistema de base de datos de elementos gestionados. Así, surgieron **SMI (*Structure of Management Information*)** y **MIB (*Management Information Base*)**, definidas por primera vez en los RFCs 1065 y 1066 respectivamente [22] [23]. SMI define las convenciones básicas de formato para los objetos, mientras que las MIB definen el esquema de estos objetos. Estas dos identidades de base de datos tendrían una enorme facilidad para realizar el cambio a OSI, dado que únicamente debería cambiar

el protocolo y el software soportado [18]. Estos elementos aún son empleados, con sus respectivas evoluciones, en las versiones actuales de SNMP.

Todos estos hechos hacían pensar en un futuro cercano provechoso para este novedoso protocolo. El tiempo demostró que el porvenir que le deparaba a SNMP no se basaba únicamente en el futuro cercano, ya que su uso iría de la mano con el de la pila de protocolos TCP/IP.

2.1.3. Evolución y distintas versiones

Libre de las restricciones ligadas a la compatibilidad con OSI, SNMP progresó rápidamente, siendo implementado casi desde su inicio por la mayoría del mercado. Como hemos dicho antes, no se puede entender la evolución de SNMP sin la evolución de TCP/IP, y la fortuna para este protocolo quiso que la entrada en escena del modelo OSI nunca se produjera.

Cada versión de SNMP ha venido a suplir deficiencias de la anterior o las anteriores, o a aportar nuevas funcionalidades. Así pues, recorramos la historia de éstas.

SNMPv1

La primera versión oficial de SNMP, publicada en el RFC 1067 en agosto de 1988, evoluciona de SMGP. Se nutre de este protocolo, tomando como buenas las tres operaciones que mencionamos en la sección 2.1.1, (léase *get, trap y set*), evolucionando a partir de aquí.

Tal vez, el mayor avance sea la **incorporación de las convenciones SMI y las MIB** para el tratamiento de la base de datos, pero no es el único avance de esta primera versión con respecto a SGMP. Por ejemplo, SGMP definía cinco tipos de «trampas» o *traps*: *Cold Start, Warm Start, Link Failure, Authentication Failure* y *EGP Neighbor Loss*. A estos cinco tipos, que dotaban a SGMP de un mecanismo de notificaciones nada desdeñable para su época y para las necesidades existentes, SNMPv1 introduce un nuevo tipo de notificación, al tiempo que reformula el tipo *Link Failure* para convertirlo en *linkDown* y, en caso de activación de nuevo de ese enlace, *linkUp*. La nueva «trampa» introducida se trata del *enterpriseSpecific Trap*, destinada a que los distintos fabricantes introdujeran sus tipos específicos de notificaciones.

Esta *trap* será lanzada cuando un evento del fabricante ocurra. El hecho que ocurra será notificado en el campo *specific-trap* [21]. Así mismo, esta primera versión introduce el concepto de *GetNext*, que será tratado específicamente en la sección 2.2.

Dentro aún de la versión 1, el RFC fue reformulado por el RFC 1098 en abril de 1989, para incluir las recomendaciones relacionadas por la IAB con respecto al uso de SNMP y CMOT explicadas en la sección 2.1.1 [24], y por el RFC 1157 para realizar correcciones tipográficas menores [25].

SNMPv2

La versión 2 del protocolo SNMP, aún en amplio uso a día de hoy, hace su aparición en abril de 1993, con el conjunto de RFCs 1441-1452. Trae consigo importantes cambios en cuanto a la autenticación (basada en *communities* en la versión 1) y la seguridad de las comunicaciones, especificadas especialmente en el RFC 1446 [5].

Además, va a apoyarse en los cambios previos de gestión de los datos, con la segunda versión de MIB [6] y la coetánea segunda versión de SMI [26]. De hecho, estas son versiones de largo recorrido pues, aunque SMI sigue empleándose en su versión segunda con una última actualización de 1999 [27], MIB-II se utiliza tal cual fue definida en la inmensa mayoría de las implementaciones de SNMP.

Las citadas modificaciones en cuanto a la seguridad no convencieron al mundo de la administración de red, que consideraba que con ellas, SNMP perdía su simplicidad inicial, prefiriendo el uso de las *communities* al de los mecanismos de autenticación propuestos en el RFC 1446 [28]. Es por ello que esta versión del protocolo es reformulada en 1996 en dos nuevas «sub-versiones», **SNMPv2c**, con una autenticación basada en *communities* y **SNMPv2u**, basada en usuarios.

El regreso de la autenticación basada en *communities* produjo una amplia aceptación de SNMPv2c, mientras que la autenticación basada en usuarios propuesta por SNMPv2u no ganó demasiados adeptos.

SNMPv2c es la formulación de SNMPv2 ampliamente usada hoy día. No obstante, las deficiencias de seguridad son evidentes, desde el momento en que la autenticación se basa en *communities*, almacenadas y transmitidas en

texto plano. SNMPv2c delega inicialmente la seguridad en otros dispositivos o técnicas, como el uso de *firewalls* o VLANs (*Virtual Local Area Network*) [3].

Es precisamente esta inquietud por la seguridad la que motiva la aparición de la tercera versión del protocolo, en el año 1998 [29].

SNMPv3

La última modificación de esta tercera versión, cuyo cambio principal es la adición de seguridad criptográfica al protocolo, amén de distintos cambios de conceptos y terminología, data de diciembre de 2002, con el conjunto de RFC 3411-3418.

Aunque no se introducen nuevas operaciones, el equipo de desarrollo de esta versión intentó hacer que todo pareciera diferente, haciendo que uno pudiese pensar que las operaciones eran totalmente distintas debido a estos cambios terminológicos [3].

La versión 3 de SNMP introduce los siguientes cambios importantes para la seguridad: un intercambio de clave **Diffie-Hellman** aplicado a SNMP y plasmado en el RFC 2786 [30] y los RFCs 3414 y 3415, que detallan los nuevos mecanismos de seguridad, **USM (*User-based Security Model*)** y **VACM (*View-based Access Control Model*)** [31,32], de los que hablaremos con más detenimiento en la sección 2.2.5.

2.1.4. Uso actual

Y llegados a este punto, cabría preguntarse si aún tiene vigencia un protocolo que ha adolecido problemas de seguridad en sus primeras versiones y del que la última versión, que trata de solventar este problema, data de hace más de quince años.

Lo cierto es que su alto grado de penetración en todo el mundo de la gestión de red desde un primer momento hace que hoy en día resulte difícil pensar en una red sin SNMP. Novedosas tecnologías, como aquellas destinadas a la SDN (*Software Defined Network*) llevan consigo implementaciones de gestión y monitorización basadas en el protocolo SNMP, como ocurre con el caso de *Nuage Networks* en Nokia [11] o SD-WAN (*Software Defined-Wide Area Network) Viptela* para Cisco [33].

Además, a lo largo de los años han ido surgiendo diversas especificaciones para favorecer que nuestro protocolo perdure en el tiempo, siendo las relativas a la seguridad las más importantes. Así, en enero de 2004 se definió el uso de AES (*Advanced Encryption Standard*) conjuntamente con SNMP [34], en enero de 2009 se publicó la integración de SSH (*Secure SHell*) para SNMP [35], en agosto de 2009 se incluyó RADIUS (*Remote Authentication Dial-In User Service*) a la lista de protocolos de seguridad integrables con SNMP [36] y en julio de 2011 fue TLS (*Transport Layer Security*) quien se sumó a la lista de protocolos de seguridad compatibles con SNMP [37]. Y las actualizaciones de seguridad no son las únicas para este protocolo ya que, a lo largo de estos quince años, se han ido añadiendo numerosas posibilidades adicionales a las existentes anteriormente.

Todo este impulso al protocolo, y su potente implantación en el mercado actual, hacen que no sólo sea utilizado a día de hoy, sino que podamos esperar que el uso de este longevo protocolo se prolongue durante bastantes años más.

2.2. Características

En la sección que comienza hablaremos sobre las características que presenta el protocolo SNMP. Se pretende con ello que el lector o la lectora comprenda las funcionalidades que serán empleadas posteriormente por nuestra aplicación.

2.2.1. Introducción y características generales

Cuando uno se enfrenta inicialmente a SNMP puede pensar que el hecho de la existencia de diferentes versiones hará difícil su interoperabilidad y que decantarse por una u otra sería una decisión excluyente. La realidad es bien distinta. Como hemos mencionado en la sección anterior, cada versión de SNMP trae consigo **avances** con respecto a su predecesora, pero no cambia su manera de actuar más allá de las particularidades de cada versión. Es decir, si queremos diseñar una aplicación para modificar valores de un *switch* empleando SNMPv3, podremos aún recibir notificaciones o *traps* de este mismo elemento utilizando SNMPv1. Esta compatibilidad entre versiones nos invita a estructurar de esta manera la sección. Así, podemos dar unas características generales para redes gestionadas por SNMP, pues esto no va a

cambiar en las tres versiones del protocolo. Además, pese a existir cambios en las MIB entre versiones 1 y 2, podemos explicar su estructura de una manera general, razonando su porqué e introduciéndonos en el mundo de las distintas MIB desarrolladas por diversos fabricantes. Tras esto, nos sumergiremos en las operaciones definidas por el protocolo. Como hemos visto, los avances no destructivos nos permiten aprovechar las virtudes de las versiones previas. Así, las operaciones *set, get y get-next* se emplean prácticamente de igual modo en v1 y v2, mientras que las *traps* se reformularon en v2, siendo empleadas las de ambas versiones de manera habitual. Existen más operaciones posibles con SNMP, pero no nos sumergiremos más allá de estas, que serán las que tendrán interés en el posterior diseño de nuestra herramienta. Por último, hablaremos del aspecto más heterogéneo de SNMP: los mecanismos de autenticación y seguridad.

Entremos ya en las características generales. La **arquitectura** de una red gestionada por SNMP se compone de distintos elementos de red (*routers*, *switches* o incluso *hosts* tradicionales) que llevan un componente *software* que, en terminología SNMP, conoceremos como **agente** y uno o varios servidores para todos ellos. En distintos textos podemos encontrarnos a estos servidores referidos como **NMS** (*Network Management Stations*) [3].

Para las comunicaciones entre ellos, y aunque existe la posibilidad de realizar una comunicación sobre TCP [38], habitualmente se emplea SNMP sobre UDP. Debido a esto, **SNMP es un protocolo no orientado a conexión**. No se establece ninguna conexión para las comunicaciones entre agentes y NMS, de modo que cada intercambio es una acción independiente de las previas o futuras.

De este modo, las comunicaciones entre agente y servidor se pueden ver del modo que se representa en la siguiente figura 2.1.

SNMP utiliza los puertos bien conocidos 161 y 162 para las comunicaciones que se muestran en la figura. El agente escuchará en el puerto 161, mientras que el servidor deberá tener abierto el puerto 162 para escuchar la recepción de *traps*.

2.2.2. Manejo de la información: SMI, MIB y OID

La manera en que SNMP gestiona la información va de la mano de la **SMI (*Structure of Management Information*)**. Como comentamos en

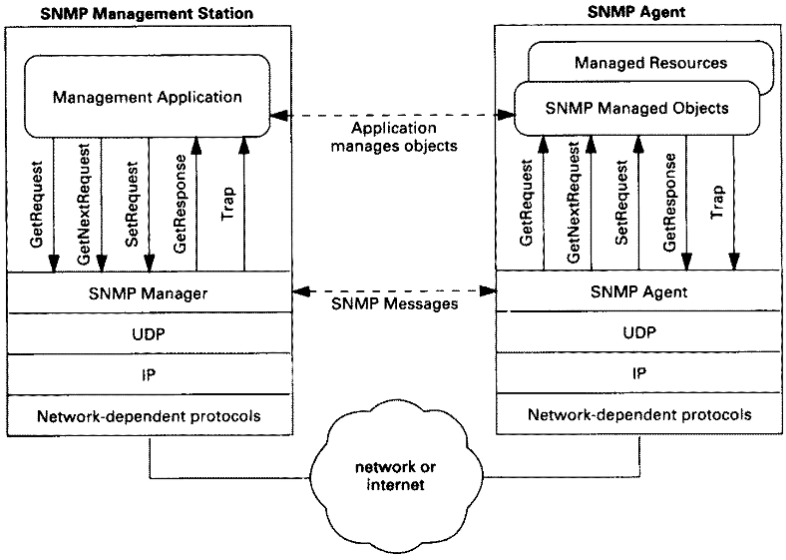

Figura 2.1: **Arquitectura y comunicaciones de SNMP** *Fuente:* [39]

la sección anterior, SMI fue desarrollada para ser usada conjuntamente por CMIP/CMOT y SNMP de cara a que una posible futura transición fuese posible.

SMI define su información en forma de «árbol». Así, se presenta ésta de una manera jerárquica, como podemos observar en la figura 2.2.

De este modo, la sintaxis utilizada por SMI para definir los datos emplea un subconjunto de **ASN.1 (*Abstract Syntax Notation One*)**. ASN.1 es una manera de especificar cómo se representan y transmiten los datos entre equipos. En nuestro caso, entre servidor o servidores y agentes en el contexto SNMP. Lo que hace apropiado su uso en SNMP es que es independiente del tipo de sistema operativo o máquina implicada en la comunicación. De este modo, un *Windows* y un *RHEL (Red Hat Enterprise Linux)* podrían intercambiarse información sin tener que preocuparse de ningún aspecto como el ordenado de los *bytes* [3].

Volviendo a la figura 2.2, podemos aclarar qué es el **OID (*Object IDentifier*)**. El OID define un objeto gestionado. Es el identificador de dicho objeto y normalmente se expresa de manera numérica y, también, con un formato más amigable para el ser humano. Así, para el ejemplo de la figura 2.2, el OID del nodo `mgmt` podríamos definirlo como 1.3.6.1.2, o bien, como `iso.org.dod.internet.mgmt`. Así, si quisiéramos obtener el OID de un no-

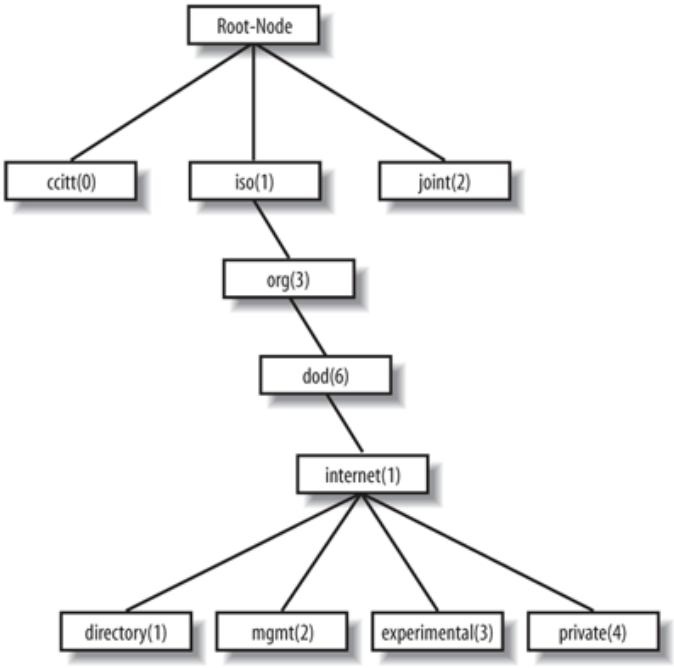

Figura 2.2: **Ejemplo de árbol SMI***Fuente:* [3]

do en concreto, lo que deberíamos hacer únicamente sería seguir la jerarquía del árbol, anexando cada índice, separado por un punto.

Conviene definir también una serie de convenciones al hablar del árbol SMI. En dicho árbol, al nodo raíz se le conoce como nodo *root*. Todos los nodos con hijos son llamados subárboles o nodos rama, mientras que los nodos sin hijos son llamados nodos hoja o nodos *leaf*.

Una vez aclarados estos aspectos, ¿qué es una «MIB»? La **MIB (*Management Information Base*)** es la base de datos de SNMP. Se apoya en SMI para disponer de un esquema de representación, como hemos visto, en forma de árbol.

Además, SMI ofrece varios tipos de datos, que son empleados por MIB. Se propusieron unos cuantos para la SMIv1 y se completaron posteriormente en SMI-II. Así, los tipos de datos existentes y una explicación de ellos se puede ver en la tabla 2.1.

Tipo de dato	**Descripción**
`INTEGER`	Número entero de 32 bits.

Tipo de dato	Descripción
`OCTET STRING`	*String* de cero o más octetos empleado generalmente para representar *strings*, pero en ocasiones usado para representar direcciones físicas.
`Counter`	Número de 32 bits con un valor mínimo de 0 y un valor máximo de $2^{32}-1$(4,294,967,265). Cuando se alcanza el valor máximo, vuelve a cero y empieza de nuevo. Su valor siempre crece, nunca decrece (excepto cuando se alcanza el tope que vuelve a 0). Es usado, por ejemplo, para determinar el número de *bytes* entrantes o salientes de una determinada interfaz.
`OBJECT IDENTIFIER`	Lista de números decimales, separados por puntos, que representa un objeto gestionado dentro del árbol MIB.
`NULL`	No usado actualmente en SNMP.
`SEQUENCE`	Define una lista que contiene cero o más tipos de datos ASN.1.
`SEQUENCE OF`	Define un objeto gestionado creado por una secuencia (`SEQUENCE`) de tipos ASN.1.
`IpAddress`	Representa una dirección IPv4.
`NetworkAddress`	Parámetro igual al anterior, pero que permite representar diferentes tipos de direcciones de red.
`Gauge`	Número de 32 bits con un valor mínimo de 0 y un valor máximo de $2^{32}-1$(4,294,967,265). Al contrario que en el caso de Counter, un Gauge puede crecer y decrecer, pero nunca puede rebasar el valor máximo. Un valor representado por Gauge es, por ejemplo, la tasa binaria de una interfaz de red.
`TimeTicks`	Número de 32 bits con un valor mínimo de 0 y un valor máximo de $2^{32}-1$(4,294,967,265). TimeTicks mide el tiempo en centésimas de segundos. El tiempo que un dispositivo lleva activo se representa en formato TimeTicks.
`Opaque`	Permite a cualquiera otra codificación ASN.1 ser almacenada como un `OCTET STRING`.
`Integer32`	Tipo introducido en SMIv2, que funciona de manera idéntica a Integer.
`Counter32`	Tipo introducido en SMIv2, que funciona de manera idéntica a Counter.

Tipo de dato	Descripción
`Gauge32`	Tipo introducido en SMIv2, que funciona de manera idéntica a Gauge.
`Unsigned32`	Tipo introducido en SMIv2, que representa un valor decimal en el rango de 0 a $2^{32} - 1$, ambos incluidos.
`Counter64`	Tipo introducido en SMIv2, similar a Counter32, pero con 64 bits, lo que permite su empleo en situaciones en que Counter32 se queda corto.
`BITS`	Tipo introducido en SMIv2, que representa una enumeración de bits.

Tabla 2.1: **Tipos de datos SMI.**

2.2.3. Envíos del agente: *Traps*

Una «trampa» SNMP es un mecanismo que tiene el agente para comunicarle al NMS que algo ha sucedido. Las *traps* fueron definidas en SNMPv1 y reformuladas en la versión 2 del protocolo, conociéndose éstas como *trap-v2* o como *notification* teniendo la misma motivación que sus antecesoras.

Existen distintas situaciones en las que una *trap* puede ser enviada. Por ejemplo, cuando una interfaz de un elemento gestionado se activa o desactiva, cuando el agente es reiniciado, cuando se produce un error de autenticación, etcétera. En la tabla 2.2 se puede comprobar de manera más detallada la lista de «trampas» genéricas.

Además, un fabricante puede definir sus propias «trampas» para sus dispositivos. Así, podría ser interesante que se enviasen *traps* por incrementos bruscos en los *bytes* entrantes o salientes por unidad de tiempo (podría sugerir un ataque o un fallo en algún punto de la red) o, en una impresora, cuando ésta se quede sin papel, o muchas otras situaciones que las «trampas» genéricas no abordan y sería interesante monitorizar.

Cuando un NMS recibe una *trap*, necesita conocer cómo interpretarla. Es decir, necesita saber qué significa la «trampa» y qué información contiene. Una *trap* es identificada por su número genérico de *trap* (mostrado en el trampa 2.2). Hay 7 *traps* genéricas. La séptima (número 6) es la usada por los fabricantes para introducir sus propias notificaciones, como hemos men-

Trap	**Definición**
`coldStart` (0)	Indica que el agente ha sido reiniciado. Todas las variables de gestión son reiniciadas. Por ejemplo, los `counters` y `gauges` son puestos a cero. Puede ser usada para determinar nuevo hardware en la red.
`warmStart` (1)	Indica un reinicio del agente pero no un reinicio de sus variables de gestión.
`linkDown` (2)	Esta *trap* es enviada cuando una interfaz del dispositivo es desactivada.
`linkUp` (3)	Esta notificación es enviada cuando una interfaz del dispositivo se activa.
`authenticationFailure` (4)	Indica que alguien ha tratado de preguntar algo a este agente con unos datos de autenticación incorrectos.
`egpNeighborLoss` (5)	Indica que se ha perdido la vecindad EGP.
`enterpriseSpecific` (6)	Indica que es una «trampa» especifica del fabricante. Para procesar este mensaje adecuadamente, el NMS debe decodificar el número específico de *trap*, que forma parte del mensaje SNMP.

Tabla 2.2: ***Traps* genéricas.**

cionado anteriormente. En estos casos, el *payload* de la «trampa» redirigirá a una MIB de este fabricante que nos servirá para decodificar la información.

Y es que es así, una *trap* se empaqueta habitualmente con información adicional. Esta información es enviada en forma de objetos MIB y sus valores, en pares que son conocidos como *variable bindings*.

En cuanto a las notificaciones, o *traps* de segunda versión, cambia el formato pero no la filosofía, y un NMS puede ser totalmente compatible para recibir, procesar y decodificar ambas. Así, sirva como ejemplo, la «trampa» *coldStart*, enviada tanto desde v1 (figura 2.3) como desde v2c (figura 2.4), y capturada mediante *Wireshark*.

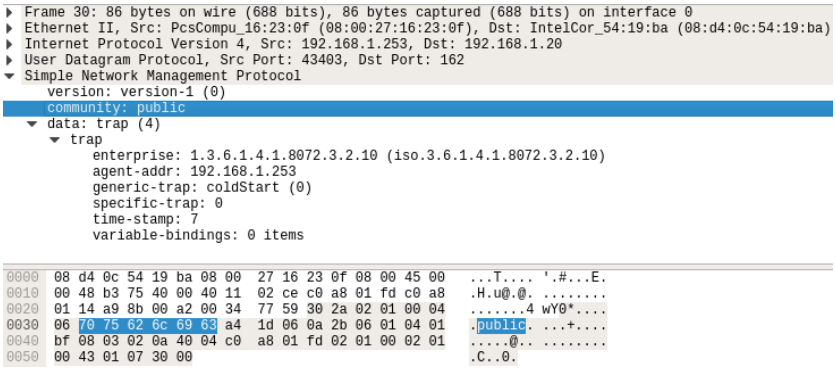

Figura 2.3: ***coldStart v1-trap.***

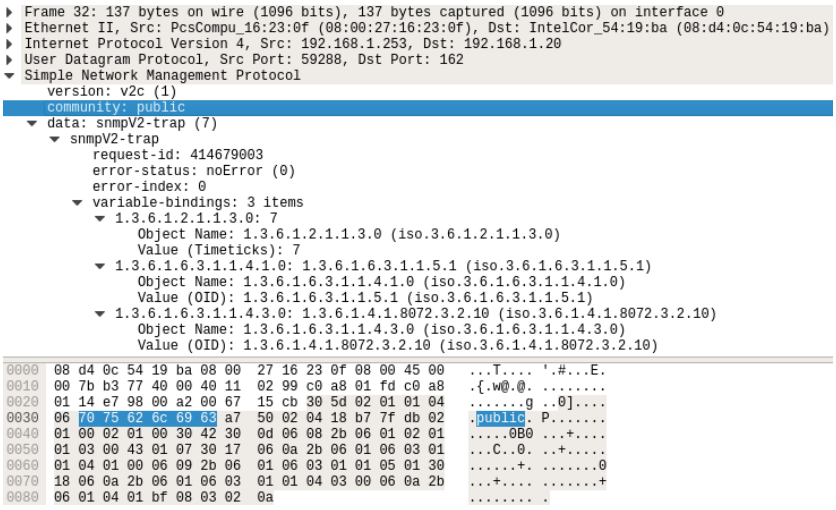

Figura 2.4: ***coldStart v2-trap.***

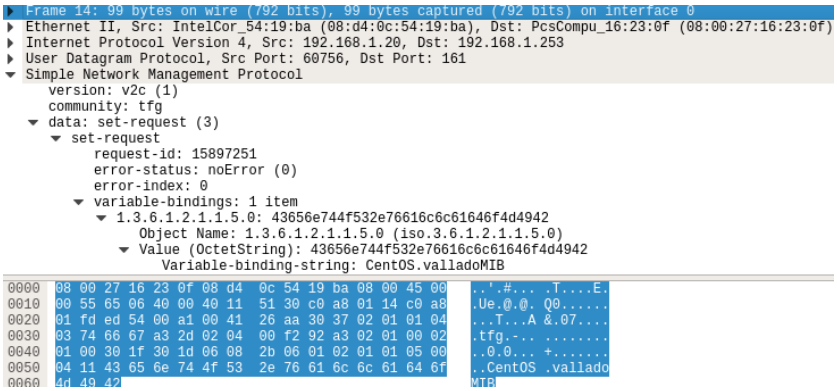

Figura 2.5: ***set-request.***

2.2.4. Envíos del servidor: *set, get, get-next*

En el apartado de los envíos que realiza el servidor tenemos varias operaciones. Nos vamos a centrar en las tres enunciadas, por considerarlas las más relevantes y las que podremos utilizar con nuestra aplicación. Empleadas conjuntamente con las «trampas», constituyen una manera eficaz, potente y versátil de gestionar uno o varios elementos de red.

set

La operación *set* es usada para cambiar el valor de un determinado objeto del agente al que se le envía la petición. Los agentes pueden ser configurados para permitir o no la modificación de ciertos parámetros, aunque algunos son obligatoriamente de sólo lectura.

La secuencia de un *set* exitoso consistiría en el mensaje enviado por el servidor, la modificación (si procediese) del valor en el agente y, por último, la notificación del resultado de la operación, enviada desde el agente al servidor. Un ejemplo de un *set* para modificar el campo *sysName* sobre v2c puede verse en las figuras 2.5 y 2.6, realizadas desde *Wireshark*.

Así pues, en la primera captura podemos ver que solicitamos el cambio de nombre a «CentOS.valladoMIB» (`Variable-binding-string`). Fijémonos también en esta primera captura en el campo `request-id`, que, en este caso, es 15897251.

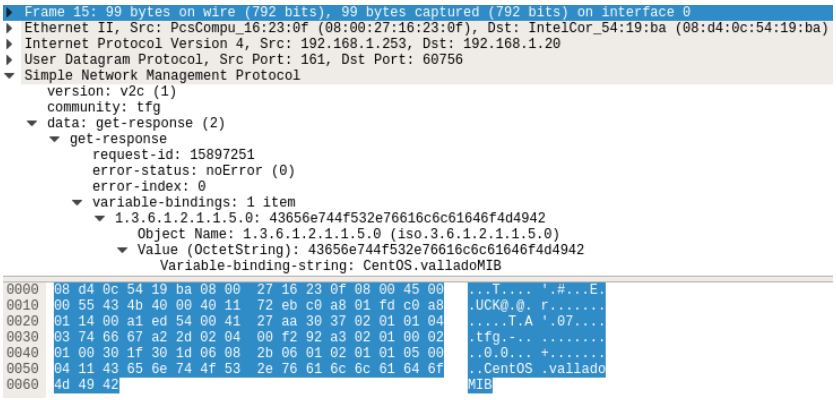

Figura 2.6: ***set-response.***

Si observamos ahora la respuesta, mostrada en la figura 2.6, con un tipo de PDU distinto (`get-response` en lugar de `set-request`), podemos ver que el `request-id` es el mismo que en la petición, lo que indica que es la respuesta al *set* que hemos enviado. Además, el `error-status` a 0, o `noError` nos indica que el cambio ha sido llevado a cabo con éxito.

Examinemos ahora los códigos de error. ¿Qué ocurriría si el cambio no hubiese podido llevarse a cabo?

Los códigos de error son definidos para las cuatro operaciones de las que vamos a hablar en esta sección y pueden verse en la tabla 2.3. Los primeros seis (códigos 0 a 5) se introdujeron en SNMPv1, mientras que los restantes fueron introducidos en SNMPv2.

Código de error	**Descripción**
`noError` (0)	La operación se ha realizado con éxito.
`tooBig` (1)	La respuesta a la petición efectuada es demasiado grande como para ser enviada en un mensaje.
`noSuchName` (2)	No se puede encontrar el OID especificado.
`badValue` (3)	El valor del *set* efectuado no concuerda con el valor que debe tener el campo objeto de la petición.
`readOnly` (4)	El objeto a alterar es de sólo lectura.
`genErr` (5)	Si el error ocurrido no puede ser agrupado en ninguno de los códigos previos, éste se engloba en `genErr`.
`noAcces` (6)	Se ha intentado un *set* a una variable inaccesible.
`wrongType` (7)	El valor del *set* efectuado no concuerda con el tipo que debe tener el campo objeto de la petición.

Código de error	Descripción
`wrongLength` (8)	Es lanzado si la entrada del *set* efectuado ha tenido que ser truncada para adecuarse a las especificaciones de longitud del objeto que se pretende modificar.
`wrongEncoding` (9)	Se ha intentado un *set* con una codificación errónea para el objeto a modificar.
`wrongValue` (10)	El valor establecido no es el correcto.
`noCreation` (11)	Se ha intentado asignar un valor a una variable no existente o se ha intentado crear una variable que no existe en el árbol MIB.
`inconsistentValue` (12)	Una variable MIB está en estado inconsistente y no está aceptando peticiones de *set*.
`resourceUnavailable` (13)	No hay recursos disponibles para realizar la operación.
`commitFailed` (14)	Si ante un *set* ocurre un error que no puede ser englobado en el resto de códigos, se envía este.
`undoFailed` (15)	El *set* ha fallado y el agente no ha podido volver a su estado original.
`authorizationError` (16)	Fallo de autorización. Este código es lanzado cuando la operación solicitada no está autorizada para completarse.
`notWritable` (17)	La variable que se quiere modificar no acepta un *set*, incluso aunque debiera.
`inconsistentName` (18)	La variable a modificar está en algún estado de inestabilidad y, por tanto, no se puede llevar a cabo la modificación.

Tabla 2.3: **Códigos de error SNMP.**

get

El propósito de la operación *get* es el de obtener el valor de un OID determinado. Es iniciada por el servidor, que envía una petición al agente, este la procesa y devuelve el resultado.

Podemos ver el intercambio de mensajes que conlleva esta operación en las figuras 2.7 y 2.8.

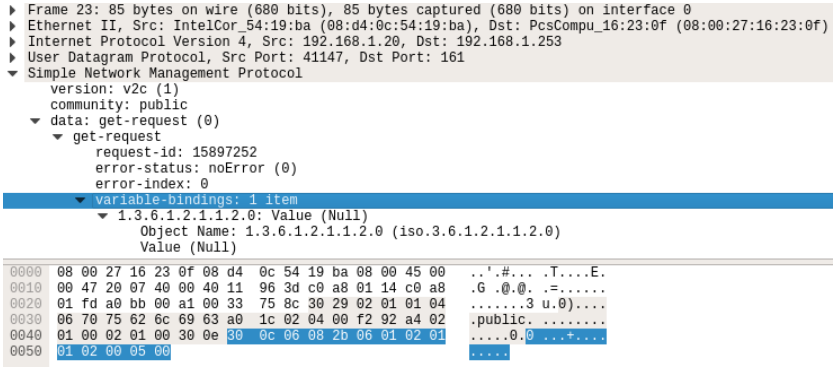

Figura 2.7: ***get-request.***

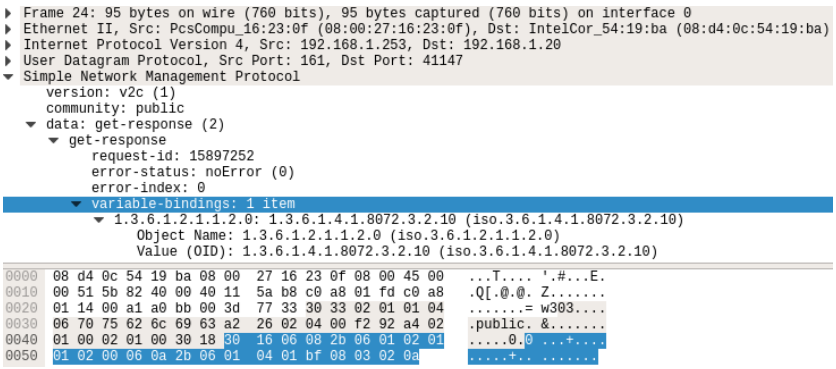

Figura 2.8: ***get-response.***

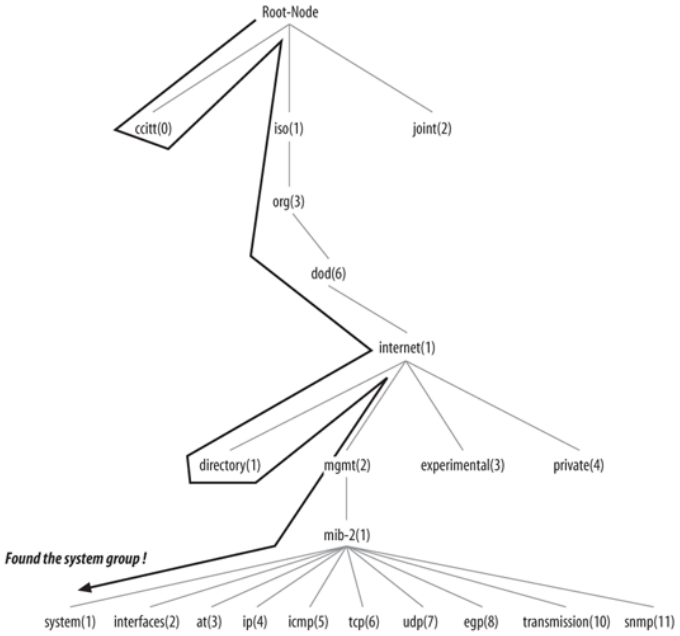

Figura 2.9: ***Walk* sobre el árbol MIB** *Fuente:* [3]

De este modo, la petición lleva consigo, en el campo `varbind`, el OID del que desea obtener valor (en este caso, `sysObjectID`, 1.3.6.1.2.1.1.2.0). La respuesta del agente, lleva en las `varbinds` este OID y su valor.

Como en el caso del *set*, el hecho de que el `request-id` sea el mismo nos indica que el `get-response` mostrado es, efectivamente, la respuesta al `get-request` inicial.

get-next

La operación *get-next* nos permite ejecutar una serie de *gets* para obtener respuesta de todo un subárbol al completo.

El comando *get-next* atraviesa el subárbol en orden lexicográfico. Teniendo en cuenta que un OID es una secuencia de enteros, resulta sencillo para un agente comenzar en el *root* del árbol completo e ir bajando hasta encontrar el OID especificado por el servidor. Un ejemplo de esta búsqueda podemos verlo en la figura 2.9. De este modo, cuando el servidor recibe la respuesta del agente, ejecuta otro *get-next*. Así se continúa hasta recorrer por completo el subárbol MIB objetivo.

```
 51 10.792426413  192.168.1.20   192.168.1.13   SNMP      83 get-next-request 1.3.6.1.2.1.1
 54 10.841151449  192.168.1.13   192.168.1.20   SNMP     137 get-response 1.3.6.1.2.1.1.1.0
 55 10.864983732  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.1.0
 56 10.869241682  192.168.1.13   192.168.1.20   SNMP      96 get-response 1.3.6.1.2.1.1.2.0
 57 10.873001106  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.2.0
 58 10.876896715  192.168.1.13   192.168.1.20   SNMP      89 get-response 1.3.6.1.2.1.1.3.0
 59 10.879015014  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.3.0
 60 10.883489730  192.168.1.13   192.168.1.20   SNMP      85 get-response 1.3.6.1.2.1.1.4.0
 61 10.885990571  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.4.0
 62 10.890415906  192.168.1.13   192.168.1.20   SNMP     100 get-response 1.3.6.1.2.1.1.5.0
 63 10.892663664  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.5.0
 64 10.896501811  192.168.1.13   192.168.1.20   SNMP      85 get-response 1.3.6.1.2.1.1.6.0
 65 10.898510760  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.6.0
 66 10.902398049  192.168.1.13   192.168.1.20   SNMP      86 get-response 1.3.6.1.2.1.1.7.0
 67 10.905650650  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.7.0
 68 10.909792327  192.168.1.13   192.168.1.20   SNMP      86 get-response 1.3.6.1.2.1.1.8.0
 69 10.912264152  192.168.1.20   192.168.1.13   SNMP      85 get-next-request 1.3.6.1.2.1.1.8.0
 70 10.916205143  192.168.1.13   192.168.1.20   SNMP      93 get-response 1.3.6.1.2.1.1.9.1.2.1
 71 10.918861595  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.2.1
 72 10.923456395  192.168.1.13   192.168.1.20   SNMP      96 get-response 1.3.6.1.2.1.1.9.1.2.2
 73 10.925821743  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.2.2
 74 10.929788174  192.168.1.13   192.168.1.20   SNMP      96 get-response 1.3.6.1.2.1.1.9.1.2.3
 75 10.931685317  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.2.3
 76 10.935883480  192.168.1.13   192.168.1.20   SNMP      96 get-response 1.3.6.1.2.1.1.9.1.2.4
 77 10.937917870  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.2.4
 78 10.944012873  192.168.1.13   192.168.1.20   SNMP      96 get-response 1.3.6.1.2.1.1.9.1.2.5
 79 10.946916143  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.2.5
 80 10.951552772  192.168.1.13   192.168.1.20   SNMP     122 get-response 1.3.6.1.2.1.1.9.1.3.1
 81 10.954527299  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.3.1
 82 10.958902975  192.168.1.13   192.168.1.20   SNMP     119 get-response 1.3.6.1.2.1.1.9.1.3.2
 83 10.961145189  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.3.2
 84 10.965236369  192.168.1.13   192.168.1.20   SNMP     125 get-response 1.3.6.1.2.1.1.9.1.3.3
 85 10.967574742  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.3.3
 86 10.971696456  192.168.1.13   192.168.1.20   SNMP      99 get-response 1.3.6.1.2.1.1.9.1.3.4
 87 10.974468844  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.3.4
 88 10.978637026  192.168.1.13   192.168.1.20   SNMP      95 get-response 1.3.6.1.2.1.1.9.1.3.5
 89 10.981992417  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.3.5
 90 10.988157369  192.168.1.13   192.168.1.20   SNMP      88 get-response 1.3.6.1.2.1.1.9.1.4.1
 91 10.991653521  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.4.1
 92 10.995395036  192.168.1.13   192.168.1.20   SNMP      88 get-response 1.3.6.1.2.1.1.9.1.4.2
 93 10.997817688  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.4.2
 94 11.001423631  192.168.1.13   192.168.1.20   SNMP      88 get-response 1.3.6.1.2.1.1.9.1.4.3
 95 11.003850342  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.4.3
 96 11.008289247  192.168.1.13   192.168.1.20   SNMP      88 get-response 1.3.6.1.2.1.1.9.1.4.4
 97 11.010667104  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.4.4
 98 11.014860457  192.168.1.13   192.168.1.20   SNMP      88 get-response 1.3.6.1.2.1.1.9.1.4.5
 99 11.017015564  192.168.1.20   192.168.1.13   SNMP      87 get-next-request 1.3.6.1.2.1.1.9.1.4.5
100 11.021814844  192.168.1.13   192.168.1.20   SNMP      86 get-response 1.3.6.1.2.1.2.1.0
```

Figura 2.10: ***get-next* sobre *SNMPv2-MIB::system*.**

Así, en la figura 2.10, podemos ver un ejemplo de uso, para el subárbol *SNMPv2-MIB::system*.

Como podemos observar, la petición inicial del *get-next* corresponde al OID 1.3.6.1.2.1.1. Ante esta petición, el agente determina el primer objeto del subárbol indicado, que en este caso lleva el OID 1.3.6.1.2.1.1.1.0 (`sysDescr.0`). Tras esto, el servidor solicita el objeto que se encuentra a continuación de `sysDescr.0` (fíjese el lector en que el mensaje ahora es `get-next-request` 1.3.6.1.2.1.1.1.0) y así prosigue hasta llegar al final del subárbol inspeccionado.

2.2.5. Autenticación: *Communities, user-based, v3*

Como venimos afirmando durante todo el capítulo, el aspecto que, probablemente, más debate ha suscitado a lo largo de la historia de SNMP es el de la autenticación. Vamos a abordar en esta sección la autenticación por ***communities***, introducida en la versión 1, no presente en la primera aproximación de la versión 2 e incluido posteriormente en la v2c; la auten-

ticación basada en usuario (***user-based***) propuesta en la versión v2u, cuyo uso es prácticamente nulo; y los mecanismos de autenticación enunciados en SNMPv3: ***User-based Security Model*** **(USM)** y ***View-based Access Control Model*** **(VACM)**.

Autenticación por *communities*

El mecanismo de autenticación por *communities*, introducido en la versión 1 sigue estando aún a día de hoy en amplio uso. Muchas organizaciones, en lugar de optar por un mecanismo distinto al simple método de las *communities*, optan por aplicar seguridad criptográfica a estas (sirvan como ejemplo los RFCs mencionados en la sección 2.1.4, como por ejemplo el destinado a utilizar TLS junto con SNMP) y por tanto, pese a la falta de seguridad que de estas se desprenden (a priori se almacenan y transmiten en texto plano) siguen siendo un aspecto relevante del protocolo hoy en día.

Las *communities* son esencialmente contraseñas. Existen diferentes grupos: sólo lectura y lectura/escritura [4]. Así, con una *community* de sólo lectura, podríamos realizar operaciones como *get* o *get-next*, mientras que con una de lectura/escritura añadiríamos a ese grupo las operaciones como *set*.

Autenticación *v2 user-based*

En el RFC 1910 se define un método de autenticación cuyo uso no llegó a penetrar con fuerza en el mercado. Este mecanismo de autenticación es conocido como *user-based*. No nos detendremos demasiado en este punto, puesto que no fue demasiado aceptado en su momento y hoy en día, aún menos, tras la publicación de la versión 3 del protocolo, que también basa su mecanismo de autenticación en usuario con el uso conjunto de USM y VACM.

En esta versión del protocolo (SNMPv2u), se define como método de acceso a los distintos objetos del agente, un nombre de usuario. Además, define una manera de realizar un *digest* del mensaje para garantizar que el emisor del mensaje realmente es quien dice ser. También proporciona un campo para indicar si los mensajes deben ser cifrados y, en caso afirmativo, el protocolo usado para garantizar la confidencialidad buscada [40].

SNMPv3: *User-based Security Model*

User-based Security Model (USM) para SNMPv3 se define en el RFC 2274. Tomando como buenas parte de las propiedades de SNMPv2u, esta especificación provee:

- **Autenticación**: Proporciona integridad de los datos y autenticación del origen de los datos. Se utiliza un HMAC (*Hash Message Authentication Code*) con función *hash* MD5 (*Message-Digest algorithm 5*) o SHA-1 (*Secure-Hash Algorithm 1*).
- **Control temporal**: Protege el mensaje contra retrasos o duplicados.
- **Confidencialidad**: Proporciona privacidad al cuerpo del mensaje empleando un cifrado de tipo DES (*Data Encryption Standard*).
- **Formato del mensaje**: Se define el formato de un campo llamado *msgSecurityParameters*, que soporta las tres funcionalidades que hemos enunciado previas a esta.
- **Descubrimiento**: Define procedimientos por los cuales un *SNMP engine* obtiene información de otro.
- **Manejo de claves**: Define los mecanismos para la generación, actualización y uso de claves.

En virtud de esto, USM opera en el envío y recepción de mensajes, determinando si el mensaje en cuestión requiere o no confidencialidad y/o autenticación y, en caso afirmativo, aplica los mecanismos de *hash* y/o cifrado necesarios. El procesamiento de estos mensajes podemos verlos en la figura 2.11.

Aunque se ha demostrado la debilidad de los mecanismos de cifrado propuestos por el RFC 2274 [41], se muestra claramente la inquietud existente por la seguridad. Así pues, USM, conjuntamente con VACM proporcionaba, para SNMPv3, de un mecanismo de autenticación, integridad de datos y confidencialidad muy importante para su época, mantenido hoy día y buscado como prioridad en muchos casos.

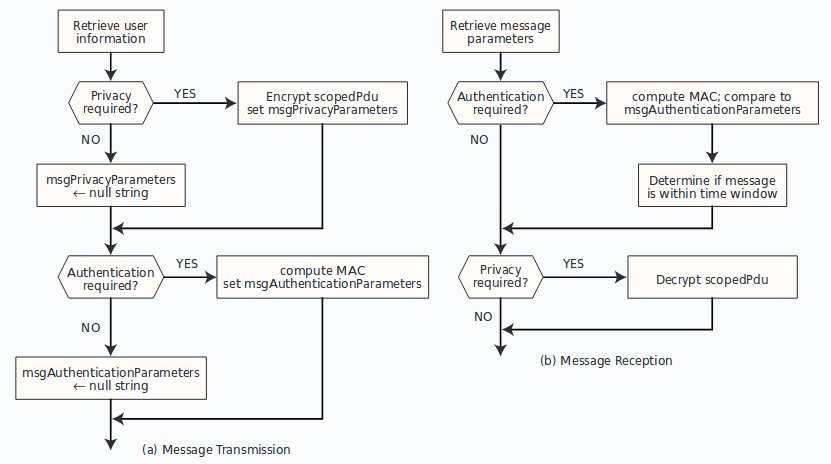

Figura 2.11: **Procesamiento de mensajes con USM***Fuente:* [4]

SNMPv3: *View-based Access Control Model*

VACM (*View-based Acces Control Model*) dispone de dos importantes características:

- VACM determina cuándo es permitido el acceso a un objeto en el árbol local MIB desde una ubicación remota. Es decir, a un agente desde un servidor.
- VACM hace un uso de MIB que define la política de control de acceso para un cliente en específico y hace posible que sea configurado de manera remota.

VACM determina el acceso o no a un OID basándose en diversos criterios. Para localizarlos, pensemos en identificar **quién** pide el acceso, **dónde** pide acceso, **cómo** lo pide, **para qué** lo pide y **a qué** pide acceso.

Si analizamos el **quién**, VACM define un ***groupName***, como un par de valores formados por el ***securityModel*** y el ***securityName***. Un *securityName* se refiere a un nombre. Todos los privilegios para este mismo nombre son idénticos, y son definidos por el *securityModel*. Cualquier combinación de *securityModel* y *securityName* puede pertenecer, como mucho, a un único grupo.

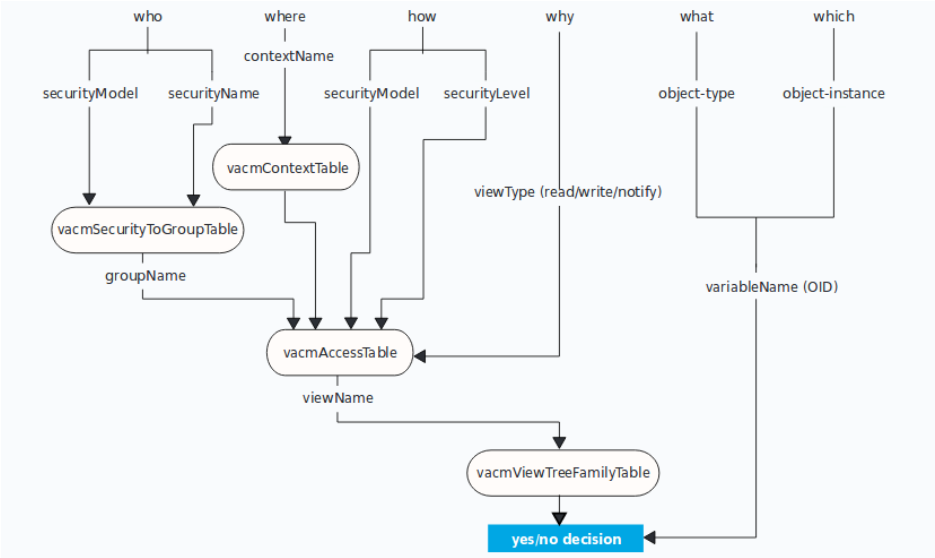

Figura 2.12: **Toma de decisiones VACM** *Fuente:* [4]

Si queremos determinar el **dónde**, VACM lo tratará ahora con el ***contextName***. Un contexto MIB es un subconjunto de instancias de objetos en la MIB local. Entonces, el atributo *contextName* se refiere a qué subconjunto se pretende acceder.

Si pensamos en el **cómo**, VACM asociará el ***securityModel*** y el ***securityLevel*** para determinarlo. El *securityModel*, mencionado pero no explicado anteriormente, define un modelo de seguridad, que puede ser, por ejemplo USM o SNMPv1. En cuanto al *securityLevel*, se refiere al nivel de seguridad requerido para una acción. Puede indicar que el mensaje ha de ser autenticado mediante algoritmos HMAC, cifrado y autenticado o que va en texto plano.

En cuanto al **para qué**, indicamos aquí si lo que se desea es leer, escribir o notificar. Esto es almacenado en un atributo denominado ***viewType***.

Por último, en cuanto **a qué** se pide acceso, nos referimos al OID, llamado por VACM ***variableName***.

En virtud de todos estos factores, VACM determina si concede o no el acceso solicitado. Un resumen esquemático puede verse en la figura 2.12.

Con todo esto, SNMPv3 provee los mecanismos de seguridad demandados por gran parte del mundo de la gestión de redes. Eso sí, debido a ello el lector

habrá notado un aumento de la complejidad del protocolo que hace que aún exista un público importante para soluciones previas como SNMPv2c, al aludir a que el protocolo, que lleva el nombre de «simple», ha perdido su sencillez.

2.3. Net-SNMP

Como mencionamos al comienzo del capítulo, el proyecto Net-SNMP ha desarrollado la implementación más común de SNMP para el mundo UNIX [7], siendo también soportada para sistemas de tipo Windows [42].

El mencionarlo aquí tiene relevancia debido a que se ha construido la aplicación precisamente sobre las bibliotecas provistas gracias a sus paquetes de *software*. También se ha configurado un agente en una máquina virtual con sistema operativo *CentOS 7*, utilizando para ello también el *software* desarrollado por este equipo. Pese a ello, nuestra parada en este proyecto será breve.

Su historia se remonta a 1992 [43], siendo la última versión desarrollada la 5.7.3 [42].

El paquete Net-SNMP ofrece una serie de herramientas de línea de comandos que, basándose en las operaciones definidas previamente en la sección 2.2 del presente documento facilitan la gestión de red basada en SNMP. Así, nos encontramos con las siguientes comandos:

- **snmpdelta**: Monitoriza el cambio en las variables SNMP.
- **snmpdf**: Monitoriza el espacio en disco en un dispositivo remoto a través de SNMP.
- **snmpget**: Efectúa una operación *get* como las definidas en la sección 2.2.
- **snmpgetnext**: Obtiene la siguiente variable en una secuencia.
- **snmpset**: Efectúa una operación *set* como las definidas en la sección 2.2.
- **snmptable**: Obtiene en forma de tabla una serie de variables SNMP.

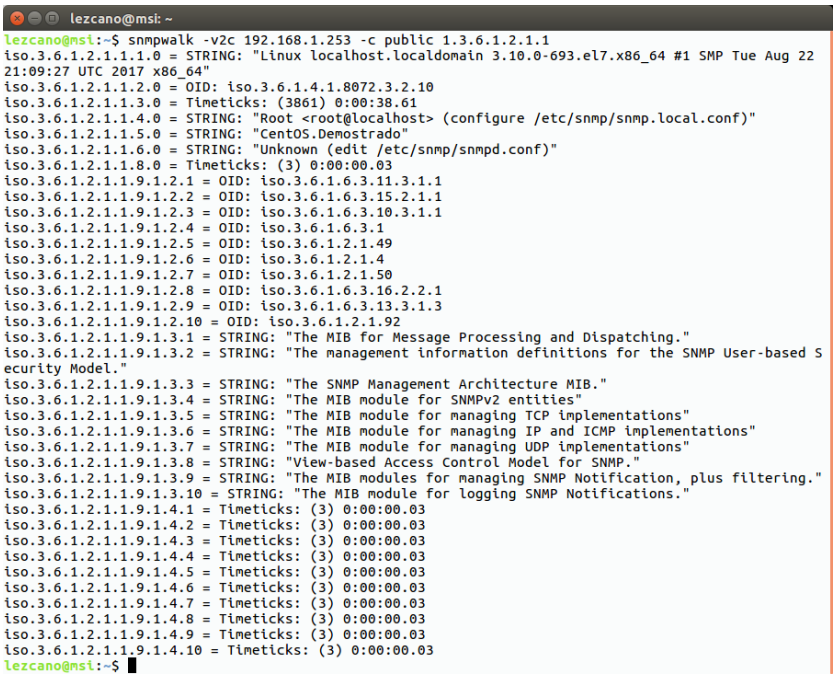

Figura 2.13: ***snmpwalk* sobre *SNMPv2-MIB::system*.**

- **snmptranslate**: Busca y describe los OIDs en la jerarquía MIB.
- **snmptrap**: Genera un *trap* como los definidos en la sección 2.2
- **snmpwalk**: Recorre por completo un subárbol MIB haciendo uso de operaciones *get-next*.

Los mismos ejemplos que se han mostrado para la sección 2.2 (realizados desde nuestra herramienta ya completamente desarrollada) podrían obtenerse vía línea de comandos gracias a Net-SNMP. Para demostrarlo, mostraremos como ejemplo el uso de *snmpwalk* sobre *SNMPv2-MIB::system*, el cual se puede ver en la figura 2.13.

Capítulo 3

Tecnologías auxiliares

Una vez visto el protocolo principal, que da motivación al Trabajo de Fin de Grado, conviene analizar, aunque sea someramente, qué tecnologías han de acompañarlo para conseguir disponer de la deseada herramienta. Para ello, en el presente capítulo, realizaremos un análisis de cuál podría ser el lenguaje de programación idóneo para este proyecto. Tras esto, hablaremos del entorno gráfico, decantándonos por aquel que nos permita resolver todos nuestros objetivos de la manera más adecuada. La tercera sección del presente capítulo hablará de la selección del sistema gestor de base de datos que utilizaremos para almacenar los dispositivos gestionados. En la cuarta y última sección mencionaremos el resto de tecnologías auxiliares que nos han acompañado a lo largo del proyecto, ya sin realizar un análisis del porqué de su elección.

3.1. Lenguaje de programación

La elección del lenguaje de programación a utilizar al desarrollar una aplicación no es trivial. Ni siquiera el paradigma elegido lo es, puesto que escoger uno u otro conlleva unas implicaciones que se pueden llegar a venir en contra del programador si no se han analizado con detenimiento las virtudes y los defectos que, lenguajes y paradigmas, puedan llegar a implicar en la escritura de la aplicación.

Hagamos primero un ejercicio de reflexión acerca de los dos grandes paradigmas de programación hoy día, el estructurado y el orientado a objetos. A

nadie se le puede pasar por alto que existen más posibilidades de desarrollo, como la orientación a eventos, pero acotemos nuestra búsqueda limitándonos a estos dos paradigmas por ser los más convenientes para este tipo de aplicación. Así mismo, convendría pensar también en si un lenguaje no compilado podría sernos de utilidad. Los lenguajes interpretados son utilizados en gran medida en ámbitos de gestión de redes, razón por la que no deberíamos pasar por alto esta opción.

Por un lado, la programación estructurada puede ser muy potente y eficiente (según qué lenguajes, más o menos) y permite la programación a distintos niveles, sin llegar a tan alto nivel como ofrece la orientación a objetos. Un ejemplo de programación eficiente utilizando la estructurada (por ejemplo, en lenguaje C) podría ser la programación de protocolos de comunicación como CAN, LIN, I2C, etcétera, donde un desarrollo basado en objetos puede, en ocasiones, significar una pérdida de prestaciones significativa en ciertos casos.

Por otro lado, la programación orientada a objetos nos abstrae de elementos de bajo nivel, eliminando ciertas complejidades a cambio, en ciertos casos, de una pérdida de eficiencia. Esta degradación podemos tolerarla en el caso de que no sea especialmente importante y la complejidad de la programación sea tal que no alcancemos el justo equilibrio entre rendimiento, prestaciones y esfuerzo dedicado para conseguirlo. Es por ello que se utilizan lenguajes orientados a objetos para el desarrollo, por ejemplo, de interfaces gráficas.

Los lenguajes interpretados se caracterizan por ejecutar directamente las instrucciones línea a línea, sin necesidad de una compilación previa. La principal desventaja de éstos (su lentitud) podría no suponer demasiado problema si las instrucciones que se tratan no son excesivamente complejas, pudiendo aprovecharnos de sus ventajas, como por ejemplo un tamaño menor (en términos de almacenamiento) o una mayor facilidad para la depuración.

Así pues, nos hallamos entre dos mares en la elección del lenguaje puesto que, además de querer trabajar con el protocolo SNMP, queremos representar los resultados de una manera gráfica, accesible y amigable para el administrador de red, cosa que puede marcarnos la elección entre uno y otro paradigma.

No obstante, a pesar de esta introducción, debemos centrar los conceptos, más o menos abstractos, expuestos en las líneas precedentes. Hablemos pues,

de las diferentes librerías que podemos encontrar para distintos lenguajes, tanto para el protocolo SNMP como para el desarrollo de la interfaz gráfica. El resultado de nuestro análisis nos proporcionará el equilibrio deseado.

Por otro lado, y a fin de simplificar nuestro análisis, centrémonos en el lenguaje **C** como baluarte de los estructurados, analicemos ***Java*** para el caso de la orientación a objetos y estudiemos ***Python*** como ejemplo de los lenguajes interpretados.

3.1.1. Librerías SNMP en los lenguajes objeto de análisis

Si nos fijamos en **C**, podemos ver que está presente en SNMP desde los inicios de este protocolo. El primer fichero de cabecera surgido, `snmp.h`, data de 1989, únicamente un año después del lanzamiento de la primera versión del protocolo [44]. No obstante, y tal y como hemos mencionado en la introducción previa, tal vez la mayor potencia alcanzable con este lenguaje respecto a otros no compense su aumento de complejidad.

Si hablamos de ***Java*** hablamos de uno de los lenguajes de programación más empleados ya no hoy en día sino desde hace más de diez años. El índice PYPL (*PopularitY of Programming Language*) lo sitúa de hecho a la cabeza en el uso de lenguajes de programación desde antes de 2005 [45]. Debido a su amplio uso, resulta indudable que existirán librerías destinadas al desarrollo con SNMP. Entre ellas encontramos `netsnmpj`, que nos permite realizar operaciones SNMP empleando el proyecto Net-SNMP [46]. También nos encontramos con otras bibliotecas, como SNMP4J, una solución bien documentada, con opciones gratuitas y otras de pago [47]. No obstante, los esfuerzos de Java en el ámbito de la gestión de red van más en la línea de JMX (*Java Management eXtensions*) [48].

En cuanto a ***Python***, nos encontramos ante un lenguaje de programación cuya proyección desde hace unos años está siendo meteórica. Si aludimos al mismo índice al que hemos hecho referencia para Java, Python suponía un uso del 2.4 % en junio de 2004, mientras que su uso se estima en torno al 22.1 % en marzo de 2018. Esta escalada contrasta con el retroceso en el uso de nuestro lenguaje estudiado, Java, que ha pasado de un máximo del 31.3 % en noviembre de 2007 al 22.6 % actual [45]. Esta evolución podemos verla en al figura 3.1, que muestra una comparativa del uso de Java, C y Python

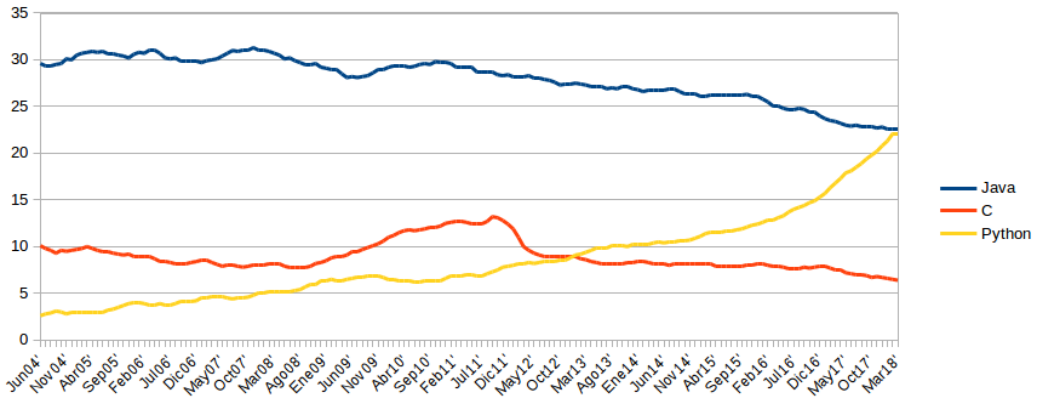

Figura 3.1: **Comparativa del uso de lenguajes de programación en el periodo 2004-2018**

Fuente: Elaboración propia a partir de los datos del índice PYPL en el periodo 2004-2018 disponibles en [45]

desde el año 2004.

Por lo tanto, como en el caso anterior, se desprendería la múltiple existencia de bibliotecas para SNMP, y así es. Partiendo de la original *Net-SNMP Python library*, se construye otra biblioteca de alto nivel conocida como *PySNMP*, un proyecto bien documentado y con múltiples facilidades para el programador, independiente de la plataforma en que se ejecute, con soporte para todas las versiones de SNMP y compatible tanto con IPv4 como con IPv6 [49].

3.1.2. Librerías gráficas en los lenguajes objeto de análisis

La disponibilidad de una interfaz gráfica adecuada será necesaria a la hora de desarrollar nuestra herramienta. Si nos fijamos en los tres lenguajes estudiados, nos encontramos con múltiples facilidades relacionadas con ***Java*** o ***Python*** con funcionalidades más bien intuitivas, mientras que, para **C**, el diseño de una interfaz gráfica medianamente compleja resulta una tarea más difícil.

El fichero de cabecera `cairo.h`, de C, podría ser empleada como complemento a la hora de diseñar algún icono, pero su uso para el desarrollo de la interfaz gráfica al completo no resulta adecuado, pues el esfuerzo para ello

quedaría sobredimensionado.

En cuanto a *Java* y *Python*, ambos lenguajes cuentan con múltiples y potentes APIs (*Application Programming Interface*) que hacen la labor del programador mucho más sencilla. Entre ellas podríamos destacar *Swing* o AWT (*Abstract Windowing Toolkit*) para *Java*, y *Tkinter*, *PyQt* o *PyGTK* en *Python*, por mencionar algunas de entre el amplio rango existente. Sin duda, ambos lenguajes se ven favorecidos en este aspecto por su amplio uso actual.

3.1.3. Discusión

Vistos los análisis previos, lo primero que podemos hacer es **eliminar de la ecuación a C**. Su complejidad, los beneficios de la orientación a objetos y el amplio uso presentado por los otros dos lenguajes generan su descarte.

Si atendemos ya a la decisión entre *Python* y *Java*, la elección no es tan inmediata. Aprovechando el paradigma de la orientación a objetos, **acabaremos descartando *Java***, principalmente por dos razones.

La primera es su falta de apuesta por SNMP como protocolo para la gestión de redes. Ante la deriva indicada en la sección 3.1.1, en que se apuesta por JMX desde *Oracle* y la existencia de costes en SNMP4J no parece una apuesta completamente recomendable. Además, *PySNMP* es una librería bien documentada, potente y de uso intuitivo, lo que provoca una selección clara en este aspecto.

Además, el aumento en la utilización de Python, que contrasta con la caída de Java, aún el lenguaje más popular en marzo de 2018, supone un argumento a favor del lenguaje diseñado por Guido van Rossum. Tanto por dejar nuestra aplicación totalmente abierta a futuras mejoras como por el propio avance que el alumno va a realizar en el que, según parece, será el lenguaje más empleado en los próximos años.

Es por ello que **elegimos Python como lenguaje de programación**. Nos apoyaremos en *PySNMP* y seleccionaremos el entorno gráfico apropiado de entre las múltiples posibilidades existentes con este lenguaje.

3.2. *PyQt* como herramienta gráfica

Una vez elegido *Python* como lenguaje de programación, debemos decidir ahora en qué nos apoyaremos para desarrollar la interfaz gráfica. Como hemos mencionado en la sección previa, existen muchas posibilidades para *Python*, desde la nativa *TkInter* a versiones híbridas con otras tecnologías como pueden ser *PyFltk*, *Jython*, *PyKDE*, *PyGTK* o *PyQt*, por citar sólo algunas del inmenso número de posibilidades.

A la hora de elegir entre todas ellas debemos tener en cuenta, ante todo, que se puede cumplir con los requisitos gráficos que nos gustaría tuviese nuestra aplicación. Por ejemplo, tras ver en qué consiste un árbol MIB, parece fundamental que nuestra aplicación pueda dibujarlo. También será necesario que soporte el manejo de tablas, para nuestros elementos gestionados. Además, dado que vamos a manejar una cantidad de líneas de código considerable, sin llegar a ser un proyecto faraónico, nos gustaría que nuestro entorno gráfico respondiese correctamente a proyectos de tamaño medio, entendiendo por tamaño medio proyectos de entre 1.000 y 10.000 líneas de código.

Con todo esto, y sin ánimo de extendernos demasiado en este aspecto, podemos ver que ***PyQt* cubre todos estos aspectos**. La clase `QTreeWidget` permite realizar representaciones en forma de árbol sin limitaciones teóricas que puedan restringirnos a la hora del desarrollo y posterior ejecución de nuestra aplicación [50]. Del mismo modo, la clase `QTableWidget` permite la representación de las tablas necesarias [51], y establecer éstas como elemento central de la ventana principal [52]. Además, resulta muy adecuado para el desarrollo de proyectos de tamaño medio como el nuestro [52]. El último elemento positivo que vamos a destacar de *PyQt* es el manejo de información y llamadas entre los distintos elementos que conforman la interfaz gráfica mediante señales, que pueden ser incluso definidas por el programador [52], lo que nos resultará útil, por ejemplo, para la recepción de las *traps*.

3.3. *SQLite* para nuestra base de datos

La existencia de datos que precisan de persistencia, más allá de aquellos almacenables gracias a las MIB, nos harán tomar una determinación acerca de la base de datos.

En nuestro programa deberemos almacenar una lista de dispositivos gestionados, con una serie de parámetros que almacenar de cada uno de ellos, como sus direcciones MAC e IP, la fecha y hora de último descubrimiento o sus *communities*. El diseño que deriva de estos requisitos comprende una única tabla, y de reducidas dimensiones, como podremos ver en el capítulo siguiente.

Teniendo en cuenta estas reducidas dimensiones, sumado además a la inexistencia de relaciones entre tablas dado que únicamente habrá una y que no van a efectuarse peticiones que no sean locales, parece evidente que el sistema gestor de base de datos a emplear debería ser lo más liviano posible y suponer para el usuario de nuestra herramienta el mínimo perjuicio posible.

SQLite nos viene como anillo al dedo para este caso. Con un biblioteca de tamaño compacto, de menos de 500 KiB [53], una arquitectura *serverless* que evita la instalación, configuración y mantenimiento de un servidor SQL (*Structured Query Language*) [54], un único fichero para la base de datos que, además, es independiente de la plataforma; es un sistema de gestión de base de datos idóneo para nuestro caso y será por lo tanto el que utilicemos.

3.4. Otras tecnologías

Además del protocolo principal, el lenguaje de programación, la utilidad gráfica y el sistema gestor de base de datos, existen otras tecnologías que han sido empleadas para el desarrollo de esta herramienta.

Entre ellas, debemos mencionar que hemos escrito el fichero de configuración con una estructura **XML (*eXtensible Markup Language*)**, dadas las facilidades ofrecidas por la bibloteca de *Python* `xml.etree.ElementTree` o `xml.etree.cElementTree`.

El entorno de desarrollo integrado (IDE, *Integrated Development Environment*) empleado ha sido ***PyCharm«®Community Edition***, mientras que se han empleado distintos protocolos de la pila **TCP/IP** tanto para apoyo como para ser parte fundamental del proyecto, basado en las necesidades de SNMP.

Todas las tecnologías mencionadas en el presente capítulo, además del protocolo objeto de estudio en el capítulo precedente, han servido para el

desarrollo de la herramienta buscada, en función del análisis y diseño que veremos en el capítulo 4.

Capítulo 4

Análisis y diseño de la aplicación

4.1. Introducción

En los orígenes de la computación, el principal esfuerzo se invertía en el desarrollo y la mejora del *hardware*, considerándose el *software* como algo accesorio, muy limitado por las restricciones físicas, quedando como resultado un desarrollo de éste «artesanal» [55].

Los resultados de este planteamiento desembocaban en un crecimiento de los costes del *software*, debido al incumplimiento sistemático de los plazos de entrega, la falta de fiabilidad del *software* construido y la imposibilidad de desarrollar aplicaciones más complejas. Estas razones provocaron el paso de la «artesanía» a la «ingeniería». La **ingeniería de *software*** podría definirse como el análisis, diseño, construcción, verificación y gestión de entidades técnicas [56].

En el presente capítulo nos dedicaremos al **análisis y diseño** de la aplicación, en el marco de la ingeniería de *software*, describiendo en el siguiente capítulo la **construcción** de la herramienta.

En el análisis, dividiremos éste en la identificación de requisitos y de casos de uso. Por su parte, a la hora del diseño, realizaremos el diseño de clases, el de la base de datos, el del fichero de configuración y el de la interfaz gráfica.

4.2. Análisis

La fase de análisis, dentro del contexto de la ingeniería de software, consiste en la recopilación de requisitos que se han de cumplir para la construcción de la posterior aplicación, así como el de situaciones deseables que deberían ser abordables por ésta.

Etapas tradicionales de esta fase pueden comprender los documentos de visión del sistema, el análisis de requisitos, el análisis de casos de uso, el análisis del modelo de dominio, así como la determinación de las secuencias y contratos de operaciones del sistema [55], aunque hay autores que separan la recopilación de requisitos del análisis propiamente dicho [57]. En este documento trataremos, para el análisis, de la recopilación de requisitos y casos de uso, por considerarlos los más relevantes para el posterior diseño y construcción de la aplicación.

4.2.1. Análisis de requisitos

En la presente sección, realizaremos un análisis de los requisitos que debería satisfacer nuestra aplicación. Para ello, dividiremos estos requisitos en funcionales y no funcionales, asignándoles un identificador por el que podremos referirnos a ellos posteriormente. Tras este análisis, se estudiarán los diferentes casos de uso y se diseñará la aplicación, buscando cumplir con todos y cada uno de los requisitos enunciados a continuación.

Requisitos funcionales

Un requisito funcional es una característica requerida del sistema que expresa una capacidad de acción del mismo, cómo debe reaccionar a determinadas entradas y cómo debe comportarse ante determinadas situaciones [58]. En este aspecto, hemos recopilado hasta 5 requisitos funcionales que nuestra aplicación debería cumplir, detallados en la tabla 4.1.

Identificador	**FRQ-001**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	10/02/2018

Descripción	El sistema deberá permitir al usuario establecer los valores que considere oportunos para aquellos OIDs de lectura/escritura.
Identificador	**FRQ-002**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá permitir la notificación de eventos por parte de los elementos gestionados.
Identificador	**FRQ-003**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá permitir realizar búsquedas entre la base de datos de elementos gestionados.
Identificador	**FRQ-004**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá permitir obtener datos de los clientes a petición del usuario.
Identificador	**FRQ-005**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá guardar en un fichero su configuración. Este fichero será leído cada vez que el sistema sea arrancado.

Tabla 4.1: Requisitos funcionales.

Requisitos no funcionales

Un requisito no funcional es una característica requerida del sistema, del proceso de desarrollo o del servicio prestado, que señala una restricción del mismo [58]. Al igual que en el caso anterior, en este aspecto, hemos podido identificar un total de 6 requisitos no funcionales, que son detallados en la tabla 4.2.

Identificador	**NFRQ-001**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá tratar las notificaciones de los equipos gestionados y mostrar al usuario alarmas en función de estos eventos.
Identificador	**NFRQ-002**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá poder listar la información correspondiente a los equipos gestionados en forma de árbol.
Identificador	**NFRQ-003**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá decodificar los OIDs a fin de presentar al usuario la información en un formato amigable.
Identificador	**NFRQ-004**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	05/02/2018
Descripción	El sistema deberá permitir al usuario alterar el fichero de configuración de la aplicación. El acceso a dicho fichero será proporcionado a través de la interfaz gráfica.
Identificador	**NFRQ-005**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	11/02/2018
Descripción	El sistema deberá permitir mostrar información detallada de un cliente concreto a petición del usuario.
Identificador	**NFRQ-006**
Autor	Gonzalo Lezcano Hermoso
Fecha de última revisión	10/05/2018
Descripción	La versión a utilizar será v2c.

Tabla 4.2: Requisitos no funcionales.

4.2.2. Análisis de casos de uso

A lo largo de la subsección que comienza, realizaremos un análisis de los casos de uso como un paso más en la ingeniería de *software* que nos llevará a la construcción de una aplicación que nos permita satisfacer los requisitos enunciados en la subsección previa. Identificaremos los casos de uso, describiéndolos, en un primer momento, de manera textual, para, posteriormente realizar su representación por medio de diagramas UML (*Unified Modelling Language*), apoyándonos en la herramienta CASE (*Computer Aided Software Engineering*) *Umbrello*.

Descripción textual de los casos de uso

ID	CambiarValores.
Satisface	FRQ-001 y NFRQ-006.
Prerrequisitos	El usuario ha hecho doble *click* sobre un elemento en la lista de elementos gestionados y ha seleccionado posteriormente la opción «Cambiar parámetros», introduciendo OID y valor y pulsando en el botón «Ok». El usuario ha seleccionado un valor de la representación en forma de árbol de un elemento, ha cambiado su valor y ha pulsado en el botón «Ok».
Elementos implicados	GUI, controlador central, base de datos, *snmpserver*, agente de elemento objetivo.
Escenario principal de éxito:	
1.	La GUI recoge la petición del usuario y la traslada al controlador central.
2.	El controlador central llama al *snmpserver* con los parámetros de la petición del snmpset.
3.	El *snmpserver* realiza la petición al elemento solicitado.
4.	El agente devuelve al servidor el resultado de la petición.
5.	El *snmpserver* informa al controlador central del resultado de la operación.
6.	El controlador central traslada el resultado (éxito o fracaso, en cuyo caso con código de error) a la GUI.
7.	La GUI presenta al usuario el resultado de su acción.

Tabla 4.3: Caso de uso CambiarValores.

ID		EnviarAlarma.
Satisface		FRQ-002 y NFRQ-001.
Elementos implicados		GUI, agente, *snmpserver*, controlador central.
Escenario principal de éxito:		
1.	El agente SNMP enviará una *trap* al servidor, en función de que se dispare alguna de las condiciones para el envío de dicha notificación.	
2.	El servidor SNMP, ante la llegada de un *trap*, reenviará al controlador central los parámetros relevantes del mismo.	
3.	El controlador central reenvía la información recibida a la GUI, añadiéndole una marca temporal.	
4.	La GUI muestra al usuario la llegada de la «trampa» en formato tabla, con el elemento que la envía, el contenido y la fecha y hora introducidas por el controlador.	

Tabla 4.4: Caso de uso EnviarAlarma.

ID		BuscarElemento.
Satisface		FRQ-003.
Elementos implicados		GUI, controlador central, base de datos.
Escenario principal de éxito:		
1.	El usuario seleccionará la opción "buscar", pulsando sobre el icono de la lupa en el menú superior de la pantalla.	
2.	Se le mostrará un desplegable en el que podrá elegir en función de qué quiere buscar, que tendrá, como primer ítem a la izquierda el desplegable, segundo ítem, en el centro, una celda de petición de datos al usuario y como tercer ítem, a la derecha, un símbolo -".	
3.	El usuario podrá añadir más campos a la búsqueda pulsando en el -"que le aparecerá en la parte derecha del desplegable.	
3.a.	En este caso, la GUI mostrará otra fila idéntica a la anterior. El usuario podrá filtrar hasta por 8 parámetros.	
3.b.	En el caso de selección múltiple, tras pulsar en siguiente, la GUI mostrará al usuario una ventana emergente, en la que deberá seleccionar si desea la búsqueda como AND lógica o como OR lógica.	
4.	La GUI pasará al controlador central los parámetros de búsqueda, incluyendo si esta es AND u OR lógica en caso de selección múltiple.	
5.	El controlador central realiza la consulta a la base de datos.	
6.	La base de datos devuelve al controlador los resultados de la petición.	

7.	El controlador central entrega estos resultados a la GUI.
8.	La GUI muestra los resultados de manera amigable al usuario.

Tabla 4.5: Caso de uso BuscarElemento.

ID		ResincronizarTablaElementos.
Satisface		FRQ-004 y NFRQ-006.
Elementos implicados		GUI, controlador central, *snmpserver*, agentes, base de datos.
Escenario principal de éxito:		
1.	El usuario seleccionará la opción «Resincronizar» del menú de opciones de la parte superior de la pantalla.	
2.	La GUI informará al controlador central de que el usuario ha solicitado una resincronización completa.	
3.	El controlador central lee del fichero de configuración el rango a inspeccionar.	
4.	El controlador central procesa la lectura del fichero de configuración y determina las peticiones a realizar.	
5.	El controlador central, para el rango gestionado, llama N veces al *snmpserver*, pidiéndole, en cada ocasión, un *get* del *sysObjectID* (.1.3.6.1.2.1.1.2.0) del elemento i (con $0 \leq i \leq$ N).	
6.	Los agentes presentes en el rango dado, responderán a dicho *get* con su *sysObjectID*.	
7.	El *snmpserver* traslada al controlador central las respuestas.	
8.	El controlador central procesa las respuestas devueltas por el *snmpserver* y prepara las nuevas peticiones al *snmpserver*.	
9.	El controlador central, por cada elemento descubierto, realiza una petición al *snmpserver* para que obtenga los parámetros relevantes que serán mostrados al usuario (*SNMPv2-MIB::system*).	
10.	El *snmpserver* realiza la consulta sobre cada elemento indicado por el controlador central.	
11.	El *snmpserver* devuelve los datos recogidos por el *snmpwalk* al controlador central.	
12.	El controlador central procesa el resultado, escribe los nuevos descubrimientos en la base de datos y actualiza los parámetros necesarios de los elementos ya existentes previamente.	
13.	El controlador central pasa los parámetros a la GUI para que sean presentados al usuario.	
14.	La GUI presenta los datos actualizados al usuario.	

Tabla 4.6: Caso de uso ResincronizarTablaElementos.

ID		AplicarConfiguración.
Satisface		FRQ-005.
Prerrequisitos		El usuario ha seleccionado la opción de aplicar configuración de la manera descrita en el caso de uso "Modificación de la configuración". La aplicación acaba de arrancar.
Elementos implicados		Fichero de configuración, controlador central. Tentativamente GUI y/o snmpserver.
Escenario principal de éxito:		
1.	Ante uno de los eventos dado como prerrequisitos, el controlador central lee el fichero de configuración.	
2.	El controlador central extrae los parámetros relevantes del fichero de configuración.	
3.	El controlador central aplica sobre los elementos adecuados (GUI o *snmpserver*) la configuración leída desde el fichero de configuración.	
4.	El elemento gestionado aplica la configuración ordenada por el controlador central.	
5.	El elemento gestionado devuelve un código de error o éxito al controlador central.	
6.	El controlador central informa a la GUI del resultado de la operación.	
7.	La GUI muestra al usuario, si procede, el resultado de la operación. Si es un error, mostrará el código de error y una posible explicación para dicho error.	

Tabla 4.7: Caso de uso AplicarConfiguración.

ID		GenerarInforme.
Satisface		NFRQ-002 y NFRQ-005.
Elementos implicados		GUI, controlador central, *snmpserver*.
Escenario principal de éxito:		
1.	El usuario seleccionará un elemento gestionado de la ventana de elementos gestionados, presentada por la GUI, pulsando con doble click sobre él.	
2.	La GUI mostrará al usuario un diálogo con una lista de opciones, entre las que tendrá las opciones de realizar un *walk* completo o de uno de sus módulos.	
3.	El usuario seleccionará alguna de las dos opciones mencionadas. Si selecciona la de un módulo, deberá indicar de qué modulo desea el informe.	

4.	La GUI traslada al controlador central la petición.
5.	El controlador central solicitará al *snmpserver* un *walk* completo o de un subárbol en concreto en función de la selección del usuario.
6.	El servidor realizará el *walk* y devolverá los resultados al controlador central. Llamará al caso de uso DecodificarOID para presentar la información en el formato adecuado.
7.	El controlador central envía estos resultados a la GUI.
8.	La GUI presenta el resultado al usuario en forma de árbol.

Tabla 4.8: Caso de uso GenerarInforme.

ID	ModificarConfiguración.
Satisface	NFRQ-004 y NFRQ-006.
Elementos implicados	GUI, fichero de configuración.
Escenario principal de éxito:	
1.	El usuario seleccionará la opción .Alterar la configuración"que será mostrada como un engranaje en la parte superior de la pantalla.
2.	La GUI mostrará entonces un diálogo con una serie de campos modificables, que el usuario podrá o no modificar.
3.	El usuario modificará lo que considere oportuno y pulsará en el botón «Aplicar», sito en la parte inferior de la ventana mostrada por la GUI.
4.	La GUI validará la corrección de los campos introducidos por el usuario.
5.	La GUI escribirá la configuración en el fichero de configuración y salvará dicha configuración.

Tabla 4.9: Caso de uso ModificarConfiguración.

ID		DecodificarOID.
Satisface		NFRQ-003.
Prerrequisitos		Petición por parte de la GUI en diversos posibles escenarios.
Elementos implicados		GUI, controlador central, *snmpserver*.
Escenario principal de éxito:		
1.	El controlador central solicitará al servidor la equivalencia de un OID en formato numérico con su nombre en formato más «amigable» para el ser humano.	
2.	El servidor decodificará este OID y devolverá el resultado al controlador.	
3.	El controlador central devolverá esta información a la GUI, que la habrá solicitado previamente en múltiples posibles escenarios.	

Tabla 4.10: Caso de uso DecodificarOID.

ID		CargarMIB.
Satisface		NFRQ-003.
Elementos implicados		GUI, controlador central, *snmpserver*.
Escenario principal de éxito:		
1.	El usuario seleccionará la opción de carga en el menú superior de la pantalla. Esta opción estará identificada con su icono correspondiente.	
2.	La GUI mostrará al usuario un diálogo donde tendrá la opción de seleccionar otra fuente y, además, seleccionar distintas MIBs a importar.	
3.	El usuario realizará su inserción en formato texto y clickará en aceptar.	
4.	La GUI presentará dicho compilador o módulo al controlador central.	
5.	El controlador central solicitará al servidor la carga de dichas MIBs.	
6.	El servidor cargará los modulos o compiladores indicados por el usuario.	
7.	El servidor comunicará al controlador central el resultado de la operación.	
8.	El controlador central informará a la GUI del resultado de la operación.	
9.	La GUI mostrará al usuario el resultado de su operación en un formato amigable.	

Tabla 4.11: Caso de uso CargarMIB.

Diagrama de casos de uso

Una vez se han enunciado los diferentes casos de uso que determinarán el comportamiento de nuestra aplicación, se procede a mostrar cómo se relacionan éstos entre sí y con los distintos actores que interactuarán con nuestra herramienta: usuario y agente.

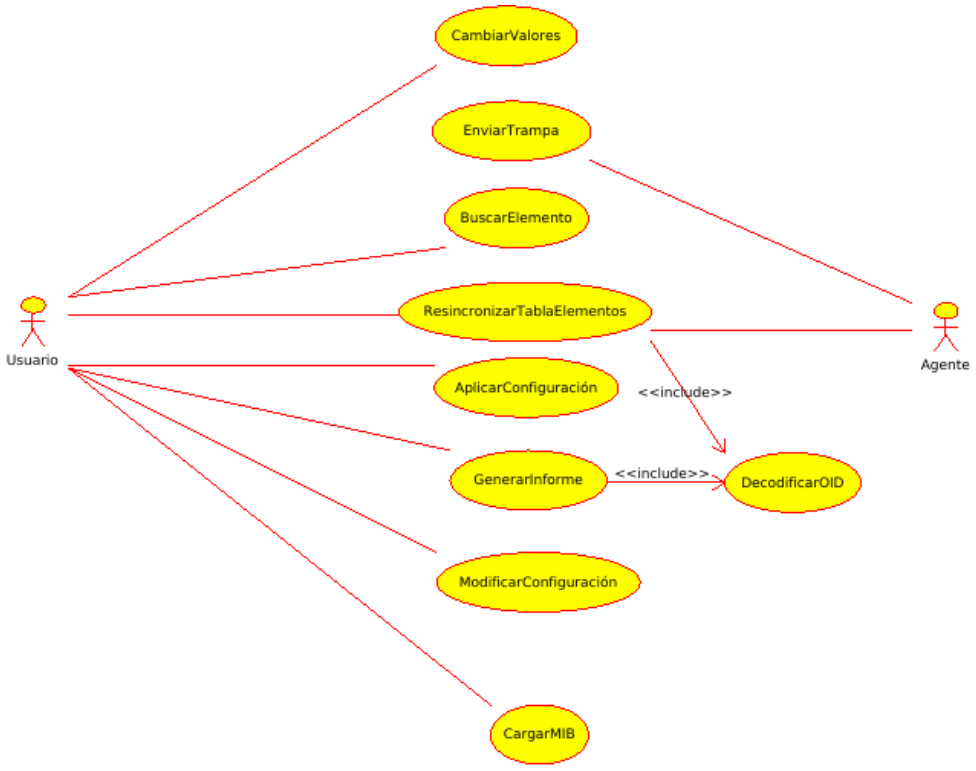

Figura 4.1: **Diagrama de casos de uso.**

4.3. Diseño

A la hora de realizar el diseño de la aplicación, en función del análisis realizado previamente y mostrado en la sección anterior, nos basaremos en el patrón modelo/vista/controlador. Este es un patrón que tiene como fin separar los datos de su representación visual y del código que los maneja.

En ella, el modelo representa el acceso a los datos, mientras que la vista presenta los datos al usuario y el controlador media entre la interfaz y el

modelo, y contiene la lógica de la aplicación.

PyQt utiliza una ligera variación de la técnica MVC (Modelo/Vista/Controlador), conocida como Modelo/Vista/Delegado, que funciona de manera similar a MVC [52]. La principal diferencia es que parte de la lógica llevada a cabo por el controlador tradicional podría ser desempeñada bien por el delegado, bien por el modelo.

Así, podemos diferenciar, basándonos en este modelo (aludiremos al controlador en lugar de al delegado), los distintos elementos que hemos ido identificando en la fase de análisis.

De este modo, la **vista** la formaría exclusivamente la **GUI (*Graphical User Interface*)**, aunque ésta comprenderá bastantes clases, como veremos algo más adelante.

Por su parte, el **controlador** sería formado por el **controlador central** y por el ***snmpserver***, que constituirán una única clase cada uno.

La parte del **modelo** vamos a dejarla para el *dataset*, en nuestro caso, la **base de datos** y el **fichero de configuración**.

Una vez realizado este diseño preliminar, metámonos en el diseño de las clases, identificándolas según hemos visto, para representar el diagrama de clases de diseño como expresión final de esta etapa. Posteriormente, también nos plantearemos el cómo en la realización del diseño de la base de datos y del fichero de configuración.

4.3.1. Diseño de clases

Identifiquemos inicialmente las clases en las que van a dividirse los diferentes elementos mencionados anteriormente.

Teniendo en cuenta que en *PyQt* cada diálogo comprenderá una nueva clase [52], la GUI, es decir, la **vista** se dividirá en:

- **MainWindow**. Representará la ventana principal.
- **EditarConfDlg**. Diálogo destinado a que el usuario pueda modificar la configuración.

- **AddModuleDlg**. Diálogo destinado a que el usuario incluya un nuevo módulo MIB al fichero de configuración para ser cargado al arranque de la aplicación.

- **LoadMibDlgIni**. Diálogo inicial para la carga de MIBs, en el que se da a elegir al usuario si prefiere cargar un módulo o un compilador.

- **LoadMibDlgModule**. Diálogo destinado a que el usuario pueda introducir un módulo MIB a cargar.

- **LoadMibDlgCompiler**. Diálogo destinado a que el usuario pueda cargar un nuevo compilador de MIBs.

- **Buscar**. Diálogo destinado a realizar búsquedas en la base de datos.

- **RepresentarResultadoBuscar**. Diálogo destinado a representar el resultado de la búsqueda del usuario.

- **SetWalkNextDlg**. Diálogo destinado a mostrar una serie de opciones cuando el usuario selecciona uno de los elementos mostrados en la tabla de elementos.

- **DlgArbol**. Diálogo destinado a mostrar una representación del árbol (o subárbol) MIB de un elemento.

Por su parte, el **controlador**, como mencionamos, estará compuesto por el controlador central y el *snmpserver*, siendo formado, por tanto, por las siguientes clases:

- **ControladorCentral**. Realizará todas las funciones destinadas al control de la aplicación. Será,fundamentalmente, un intermediario entre el servidor SNMP y la interfaz gráfica.

- **SnmpServer**. Su cometido será cumplir con todo aquello que debiera realizar un servidor SNMP. Esto es, realizar los envíos a los agentes de la manera adecuada y escuchar por la recepción de *traps*. También implementará las funciones destinadas a decodificar los OID para obtener el resultado en un formato «amigable» para el usuario y cargar módulos y compiladores MIB.

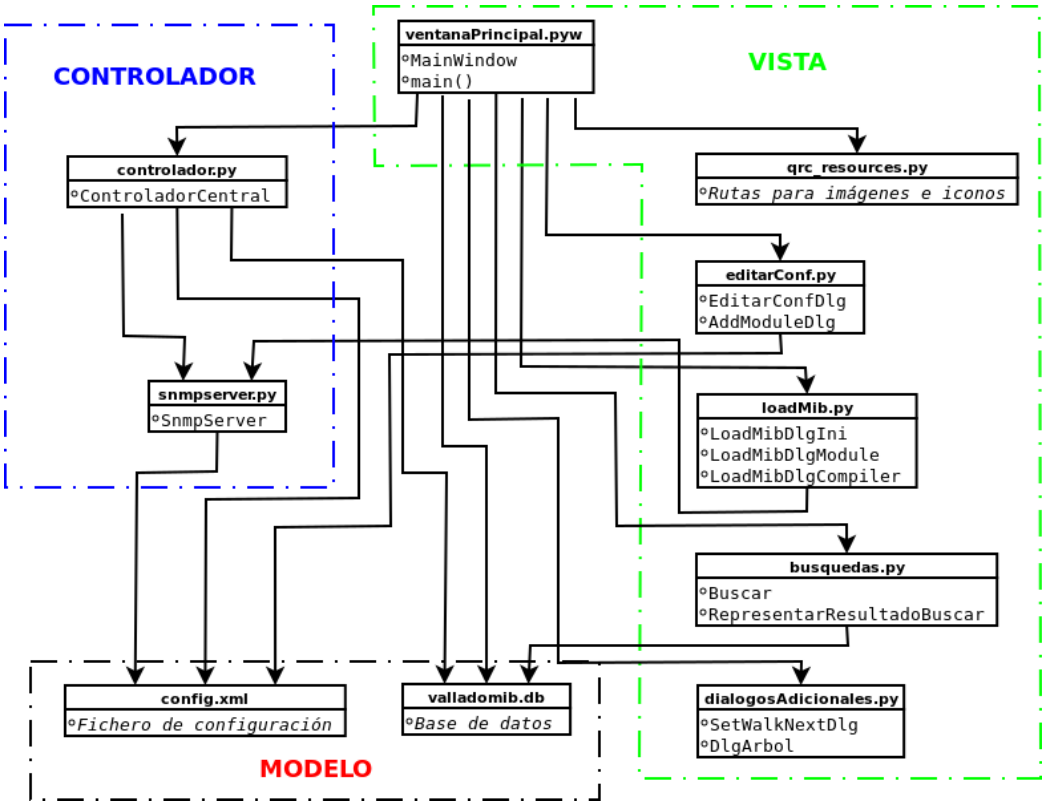

Figura 4.2: **Estructura de ficheros de la aplicación.**

En cuanto al **modelo**, y como hemos mencionado previamente, dejamos esta parte para el *dataset* y, por lo tanto, estará formado por la **base de datos** y por el **fichero de configuración**.

Así, con todo lo visto, se ha realizado una división en ficheros que podemos observar en la figura 4.2, a fin de cumplir con las especificaciones de diseño.

El fichero `qrc_recursos.py`, el único que contiene algo no mencionado previamente, proporciona una ruta inequívoca a las imágenes e iconos utilizados por el programa, independientemente del lugar desde donde sea ejecutada la herramienta ni del tipo de sistema operativo en el que se ejecute. Esto es así debido a un mecanismo de *Qt* denominado *Qt Resource System* [52]. Gracias a ello, se especifica en un fichero con extensión `.qrc` dónde se encuentran nuestros recursos. Posteriormente, se deberá realizar la conversión a código interpretable por *Python*, lo cual se consigue fácilmente, ejecutando el siguiente comando: `pyrcc4 -py3 -o qrc_recursos.py recursos.qrc`.

Nuestro fichero `recursos.qrc` tendrá el siguiente aspecto:

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
<file alias="edit.png">images/edit.png</file>
<file alias="find.png">images/find.png</file>
<file alias="resync.png">images/resync.png</file>
<file alias="config.png">images/config.png</file>
<file alias="upload.png">images/upload.png</file>
<file alias="about.png">images/about.png</file>
</qresource>
</RCC>
```

Mientras que, por su parte, el fichero *Python* resultante tendrá un formato que no será fácilmente asimilable para el ojo humano, como podemos ver en este pequeño fragmento:

```
# -*- coding: utf-8 -*-

# Resource object code
#
# Created by: The Resource Compiler for PyQt4 (Qt v4.8.7)
#
# WARNING! All changes made in this file will be lost!

from PyQt4 import QtCore

qt_resource_data = b"\
\x00\x00\x0a\xea\
\x89\
\x50\x4e\x47\x0d\x0a\x1a\x0a\x00\x00\x00\x0d\x49\x48\x44\x52\x00\
\x00\x00\xe1\x00\x00\x00\xe1\x08\x03\x00\x00\x00\x09\x6d\x22\x48\
\x00\x00\x00\x81\x50\x4c\x54\x45\xff\xff\xff\x00\x00\x00\xf6\xf6\
...
```

Una vez visto todo esto, dónde se ubicarían las distintas clases y por qué, en la figura 4.3 se muestra el diagrama de clases de diseño.

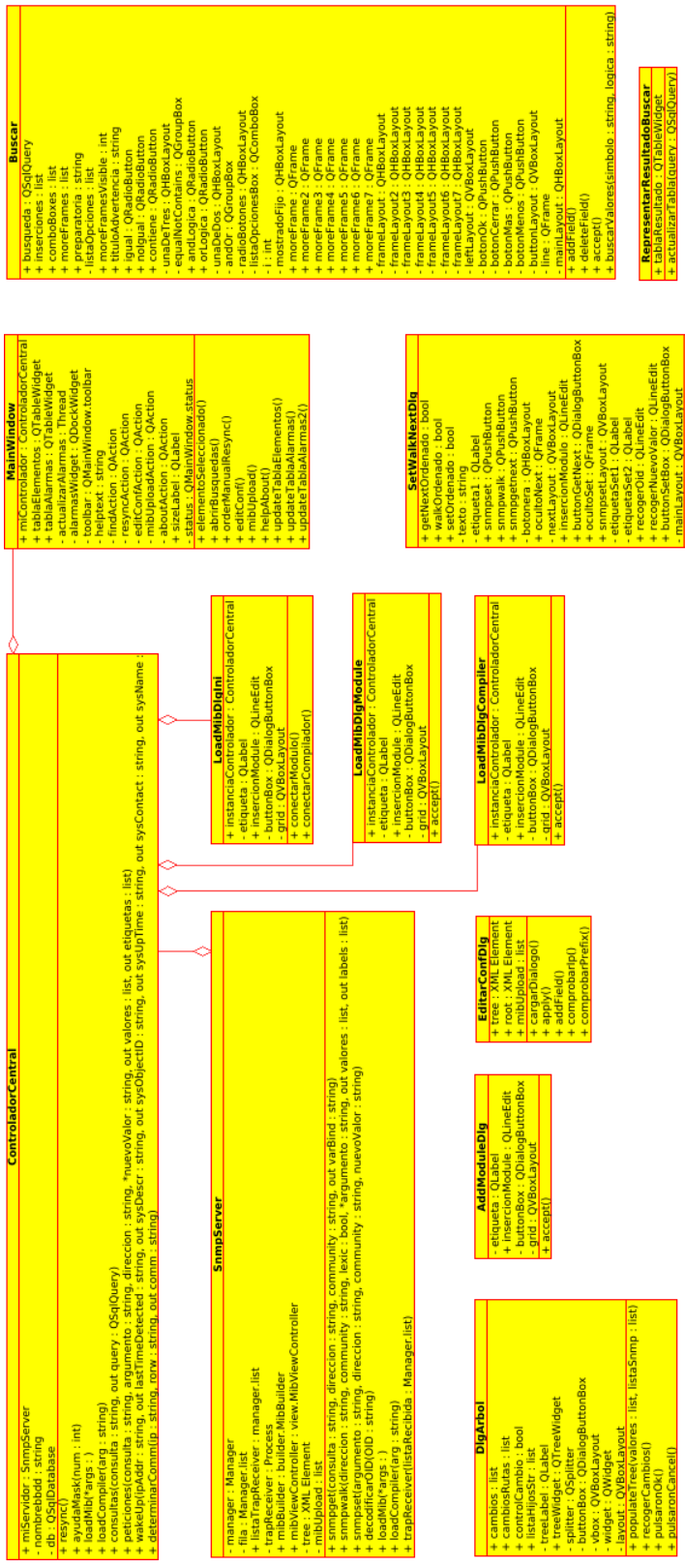

Figura 4.3: **Diagrama de clases.**

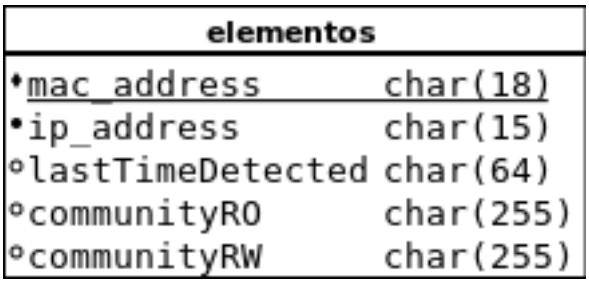

Figura 4.4: **Diseño de la base de datos.**

4.3.2. Diseño de la base de datos

A fin de dotar de persistencia a nuestro programa, se ve la necesidad de diseñar una pequeña base de datos. La motivación es la de almacenar y configurar una serie de parámetros de los dispositivos gestionables en nuestro rango de red. Para esto, como mencionamos en el capítulo 3 al elegir *SQLite* como sistema gestor de base de datos, no nos hace falta más que una pequeña tabla.

Los parámetros que deberemos almacenar son aquellos que identifican al elemento (véase dirección MAC, como clave primaria, y dirección IP), una marca horaria con el último contacto entre elemento y servidor y dos campos para almacenar las *communities* específicas del dispositivo, si existiesen.

En la figura 4.4 podemos observar cuál es el diseño final. El porqué de este diseño vamos a desgranarlo en las próximas líneas.

¿Cómo identificamos a un elemento? Lo que primero se podría llegar a pensar es en hacerlo por dirección IP, pero la posible existencia de DHCP (*Dynamic Host Configuration Protocol*) en el segmento de red gestionado y la verosímil ocurrencia de direcciones IP duplicadas desaconsejan la identificación del *host* por su dirección IP como clave primaria.

Esta es la razón por la que elegimos la **dirección MAC como clave primaria**. La decisión tiene dos ventajas principales. Facilita la gestión de problemas relacionados con la IP duplicada, pues resulta muy sencillo identificar que dos elementos con distinta dirección MAC presentes en nuestra base de datos poseen la misma dirección IP. El segundo aspecto es que facilita la gestión en una red con DHCP. Al identificar al *host* por la dirección MAC, ante un cambio de dirección IP actualizaremos dicho parámetro en la base de datos, teniendo presente su correcta correspondencia IP-MAC, identificando

inequívocamente a dicho elemento.

Es cierto que podríamos encontrarnos también ante el problema de dirección MAC duplicada, pero su aparición es mucho menos frecuente y, normalmente, para estos problemas, se dispone de mecanismos por parte de los propios fabricantes. Por ejemplo, Cisco, ante un problema de MAC duplicada, incorpora un mecanismo conocido como *MAC move*, implementado también por otros fabricantes, que consiste en que, cuando un elemento detecta que en el mismo segmento de red hay otro *host* con su misma dirección física, cambia automáticamente su dirección MAC. Cuando esto ocurre, envía una *trap* informando de ello, por lo que el usuario podría verla en la sección de alarmas y actuar en consecuencia [59].

Por otro lado, almacenaremos la fecha y hora de último contacto con formato `YYYY-MM-DD HH:MM:SS.XXXXXX`, con `Y` como año, `M` como mes, `D` como día, `H` como hora, `M` como minuto, `S` como segundo y `X` como decimal. Por ejemplo, una marca válida sería `2018-04-13 18:26:06.843321`. Se generará una marca horaria dentro del controlador en el momento necesario y posteriormente se escribirá como *string* en la base de datos. Esto permitirá al usuario determinar si realmente tiene o no capacidad de gestión del elemento que le aparece en la pantalla.

En cuanto a los campos destinados a las *communities*, permitirán determinar este mecanismo de autenticación específicamente para cada elemento.

4.3.3. Diseño del fichero de configuración

Existen ciertas opciones de configuración que no pueden ser almacenadas en nuestra base de datos sin complicar su estructura.

Por ejemplo, nos podemos encontrar con el caso de no disponer de una *community* específica para un elemento en la base de datos, debiendo disponer de una por defecto. O también, deberemos definir un segmento de red en el que realizar nuestros descubrimientos. Nótese que en este mismo punto se hace necesaria la existencia de una *community* de sólo lectura por defecto. Otro aspecto que cabría en este fichero de configuración es la carga inicial de módulos MIB. Los módulos MIB, que se pueden ver como ficheros de texto, necesitan ser compilados para que *PySNMP* pueda utilizarlos. Es por ello que, indicarle qué módulos debe compilar inicialmente también podría ser una opción interesante a incluir en este fichero de configuración.

```
−<Config>
   <Subnet>192.168.1.252</Subnet>
   <Prefix>30</Prefix>
   <CommunityRO>public</CommunityRO>
   <CommunityRW>tfg</CommunityRW>
  −<MibUpload>
     <Mib1>SNMPv2-MIB</Mib1>
     <Mib2>SNMP-FRAMEWORK-MIB</Mib2>
     <Mib3>SNMP-COMMUNITY-MIB</Mib3>
     <Mib4>IP-MIB</Mib4>
   </MibUpload>
 </Config>
```

Figura 4.5: **Ejemplo de fichero de configuración**

Así, nuestro fichero de configuración, que empleará formato XML, como ya adelantamos en el capítulo 3, tendrá 5 campos de primer nivel: `Subnet`, `Prefix`, `CommunityRO`, `CommunityRW` y `MibUpload`, siendo este último campo compuesto por cero o varios módulos MIB identificados por `Mibx`, siendo x el número de módulo MIB en orden creciente. Un ejemplo de este fichero XML podemos verlo en la figura 4.5.

4.3.4. Diseño de la interfaz gráfica de usuario

A lo largo del capítulo en que nos encontramos hemos podido ver una serie de aspectos que guían el diseño de lo que será nuestra interfaz gráfica. Por ejemplo, se ha aludido en repetidas veces a la existencia de una **barra de herramientas** con diversos iconos que nos guían a una serie de acciones que han de ser llevadas a cabo en base a los casos de uso. Así mismo, será necesario que la parte principal de la ventana principal se dedique a mostrar **información de los elementos** que se gestionan. Será necesario que sean mostrados en forma de tabla con distintos campos. Adicionalmente, y cumpliendo con nuestros requisitos, deberemos proveer de un modo de **mostrar las *traps* recibidas** por parte de los agentes. Por último, incluiremos una barra de estado.

Entonces, diseñemos nuestra pantalla principal como cuatro elementos dispuestos de forma vertical. En el primero de ellos, la **barra de herramientas**, incluiremos cinco iconos, ordenados en horizontal. Estos iconos son los siguientes:

- **Búsquedas**. Este icono nos deberá redirigir hacia un diálogo que per-

mita realizar búsquedas en la base de datos de elementos gestionados. Lo identificaremos mediante una lupa.

- **Resincronizar**. Necesario para realizar la resincronización, conocida como descubrimiento en el intervalo de red marcado en el fichero de configuración. Su misión será la de comunicar que la resincronización mencionada debe realizarse. El icono que identificará está acción estará compuesto por dos flechas que conforman un círculo entre ellas, apuntando en dirección de las agujas del reloj.

- **Editar configuración**. Lanzará un diálogo que permitirá al usuario alterar el fichero de configuración que hemos diseñado previamente. Estará identificado con un engranaje.

- **Carga de MIBs**. Debe dirigir a un diálogo que permita al usuario introducir módulos MIB o nuevos compiladores. Estará identificado con una flecha en dirección ascendente.

- **Información de la aplicación**. Al ser *clickado*, mostrará una nueva ventana con información acerca de la aplicación, como por ejemplo la versión, las tecnologías empleadas, el desarrollador, etcétera. Estará identificado mediante un «bocadillo» con una «i» dentro.

Si nos atenemos al segundo elemento mencionado, la **tabla de elementos gestionados**, deberemos encontrarnos en ella a los dispositivos que se encuentren en la base de datos, con parámetros MIB adicionales. La cabecera que deberá tener nuestra tabla estará formada, por lo tanto, por los siguientes campos:

- **Dirección MAC**. Clave primaria de la base de datos para nuestro elemento, que se pretende que lo identifique inequívocamente.

- **Dirección IP**. Dirección IPv4 del elemento en cuestión. Es leída de la base de datos.

- **Último contacto**. Fecha y hora del último contacto entre el servidor y el elemento. Se lee de la base de datos y es actualizado en el momento de dibujar la tabla si el elemento es accesible desde el servidor.

- `sysObjectID`. Este parámetro del módulo MIB `SNMPv2-MIB::system` se obtiene en el momento de dibujar la tabla, quedando en blanco si no hay respuesta por parte del elemento gestionado.

- `sysDescr`. Como en el caso anterior, obtenemos esta columna en el momento de dibujar la tabla, quedando vacía si no existe conectividad con el elemento.
- `sysUpTime`. Como en los dos casos anteriores, obtenemos este parámetro al momento de dibujar la tabla vía SNMP, si existe conectividad entre servidor y elemento gestionado.
- `sysContact`. Este cuarto parámetro obtenido gracias a SNMP y mostrado en la ventana principal sigue la misma lógica que sus tres predecesores.
- `sysName`. El `sysName` se obtiene al momento del dibujo de la tabla por SNMP, tal y como las cuatro columnas anteriores.
- `sysServices`. Será nuestra última columna. Este parámetro, como los cinco anteriores, se obtiene al momento de dibujar la tabla si existe conectividad entre servidor y elemento gestionado.

En cuanto a la tercera de las partes de nuestra ventana principal, la **tabla de alarmas**, nos mostrará información acerca de las *traps* llegadas desde los agentes durante la sesión en ejecución del programa. Para ello, dispondrá de tres columnas:

- **Elemento afectado**. Mostrará la dirección IP del elemento que envía la notificación.
- ***Trap***. Esta columna contendrá el contenido de la «trampa» enviada.
- **Llegada**. Muestra la fecha y hora de llegada de la *trap*.

La cuarta y última de las secciones de nuestra interfaz gráfica es la **barra de estado**. En ella únicamente indicaremos al usuario el progreso de ciertas acciones que vayan a requerir de una latencia relativamente elevada (como un *walk* completo) o el momento en que el programa estará listo para ser usado.

Así, con todo lo expuesto, un boceto del diseño deseado para la pantalla principal del programa puede verse en la figura 4.6.

Figura 4.6: **Boceto de la interfaz gráfica.**

Capítulo 5

Desarrollo e implementación

5.1. Introducción

Una vez finalizadas las fases de análisis y diseño toca ahora centrarnos en la etapa de **construcción** de la herramienta. Para ello, y haciendo uso de las distintas tecnologías vistas en los capítulos 2 y 3, trataremos de cubrir todos los casos de uso identificados en el capítulo 4, mediante el diseño que se describe en este mismo capítulo, en la sección 4.3.

5.2. Desarrollo de los distintos casos de uso

La sección que comienza se dedica a explicar al lector o la lectora cómo se han superado los distintos retos que se nos presentaban como casos de uso que había que desarrollar. De este modo, en un total de nueve subsecciones, cubriremos los aspectos más relevantes de la construcción de la aplicación, cuyo código puede ser examinado con mayor profundidad en los apéndices.

5.2.1. Desarrollo del caso de uso CambiarValores

Como se puede ver en la tabla 4.11, que habla de este caso de uso, hay dos situaciones que desembocan en este supuesto.

```
########################################################################
#                        Método: acceptSet                             #
#                                                                      #
# Recoge el click del usuario en el botón Ok del Set, altera el        #
# parámetro público setOrdenado y cierra el diálogo.                   #
########################################################################
def acceptSet(self):
    self.setOrdenado = True
    self.close()
```

Figura 5.1: **Desarrollo del caso de uso CambiarValores *acceptSet*.**

La primera consistirá en que el usuario, tras seleccionar un elemento de la ventana de dispositivos gestionados, elija la opción «Cambiar parámetros», introduciendo el OID deseado en formato numérico y el nuevo valor deseado para dicho objeto.

La segunda se logra tras cambiar el valor del OID que elija en una representación en forma de árbol, bien del módulo MIB elegido, bien del árbol MIB completo.

Hay que añadir que, en ambas situaciones, los diálogos que muestran, tanto el árbol, como la opción «Cambiar parámetros», son llamados desde `MainWindow(QMainWindow)`.

Tras realizar las pertinentes operaciones previas, la instancia `app` de la clase `MainWindow` inicializada en el `main()` recoge los cambios y emplea la instancia `miControlador` de `ControladorCentral` para comunicar el cambio. Es el controlador central quien se encargará de solicitar al servidor que se realice el cambio.

Esta clase (`MainWindow`) tiene un método (`elementoSeleccionado()`) destinado a recoger las posibles interacciones tras haber seleccionado un elemento de la tabla principal. Entre ellos, se encuentra la opción de realizar un `snmpset`, desde las dos opciones mencionadas previamente.

Existe un parámetro público de la clase `SetWalkNextDlg` al que llamamos `setOrdenado`. Se trata de un parámetro *booleano*, que toma el valor `True` si el usuario pulsa el botón «Ok» de dicho diálogo en su opción de «Cambiar Parámetros» y que, por defecto, toma el valor `False`. En la figura 5.1 podemos ver en qué lugar toma el valor `True`, si llega a tomarlo.

Si el cambio viene de una modificación en la representación del árbol MIB,

```
if self.controlCambio is False:
    self.controlCambio = True
```

Figura 5.2: **Desarrollo del caso de uso CambiarValores *controlCambio*.**

```
if miDlgEleccion.setOrdenado:
    self.miControlador.peticiones("set", miDlgEleccion.recogerOid.text(), ipElemento, miDlgEleccion.recogerNuevoValor.text())
```

Figura 5.3: **Desarrollo del caso de uso CambiarValores. Llamada al controlador central (1).**

se nos notifica por medio del método `recogerCambios` del diálogo `DlgArbol`, que dibuja el árbol del que estamos hablando. Esta clase tiene también un parámetro público *booleano*, llamado en este caso, `controlCambio`, puesto a `True` en el caso de existir uno o más cambios. Podemos ver el tratamiento que de este parámetro se realiza en el método mencionado, en la figura 5.2, al final del método `recogerCambios` de la clase `DlgArbol`.

En virtud de estos dos parámetros de control, se realiza, desde `MainWindow`, la llamada al controlador central, como podemos ver en las figuras 5.3 y 5.4, teniendo en cuenta que, para el caso de la segunda de estas figuras, el prerrequisito viene de la representación del árbol MIB y, por lo tanto, varios valores pueden haber sido cambiados. La manera en que se obtienen los valores del cambio, debido a que no forma parte del caso de uso, sino de un prerrequisito, no la mostraremos aquí, pero puede consultarse en el apéndice A (`dialogosAdicionales.py`).

Dentro de la clase `ControladorCentral` podemos encontrarnos ante el método `peticiones`, que toma como entradas los parámetros `consulta`, `argumento`, `direccion` y `*nuevoValor`. Se encarga de aglutinar todas las peticiones que lleguen debido a la interacción con la GUI, dirigidas al servidor SNMP. Así, `consulta` será un *string* que podrá tomar por valores *get, getnext, walk* o *set*. Nos vamos a centrar, por encontrarnos en el punto en que estamos, en el *set*.

Para realizar esta operación, necesitamos, como pudimos comprobar en el capítulo 2, y dado que estamos implementando SNMPv2c, la *community* de lectura/escritura. Para averiguarla, el método `peticiones` se apoya en otro método de la clase `ControladorCentral`, llamado `determinarComm`. Este método toma como entradas la dirección IP del elemento sobre el que se va a realizar la operación y la *community* deseada. Nuestra petición tomaría la siguiente forma:

```
# miArbol.controlCambio nos informa sobre si algún elemento del árbol ha sido modificado.
# En ese caso, intentamos un snmpset sobre el o los OIDs cuyo valor se haya modificado.
# Obtenemos la ruta como lista de etiquetas, así que la parseamos para que sea una cadena
# con las etiquetas separadas por puntos, de forma que sea interpretable para PySNMP. Así,
# para sysName, enviaríamos al controlador lo siguiente:
# 'iso.org.dod.internet.mgmt.mib-2.system.sysName.0'
if miArbol.controlCambio:
    oid = ""
    i = 0
    for cambio in miArbol.cambios:
        for nodo in miArbol.cambiosRutas[i]:
            if oid == "":
                oid = oid + nodo
            else:
                oid = oid + "." + nodo
        self.miControlador.peticiones("set", oid, ipElemento, cambio[1])
        oid = ""
        i += 1
```

Figura 5.4: **Desarrollo del caso de uso CambiarValores. Llamada al controlador central (2).**

```
communityrw = self.determinarComm(direccion, 'communityRW')
```

Esto es así dado que la dirección del elemento ha sido enviada como parámetro al método `peticiones`.

El método `determinarComm(ip, rorw)`, realiza una consulta a la base de datos para determinar si el elemento en cuestión dispone o no de una *community* específica y, en caso de no ser así, lee el fichero de configuración para determinar la *community* por defecto. También trata el posible problema de IP duplicada, no realizando el cambio si esto sucediese. Este método puede verse en la figura 5.5.

En caso de que no se obtenga *community* en el método mostrado, se impedirá la petición, como podemos ver en la figura 5.6.

Una vez obtenida la *community*, conociendo la IP del elemento sobre el que realizar la petición (y sabiendo que no nos encontramos ante un problema de IP duplicada), el OID a modificar y el nuevo valor (parámetros de entrada del método), procedemos a solicitar al servidor la operación, como podemos observar en la figura 5.7.

El método `snmpset` de `SnmpServer`, toma como entradas los parámetros `argumento`, `direccion`, `community` y `nuevoValor`.

Utiliza `argumento` como OID sobre el que realizar el cambio, `direccion` como dirección IP del agente objetivo, `community` como, precisamente, *com-*

```
######################################################################
#                  Método: determinarComm                            #
#                                                                    #
# El presente método se encarga, dada la IP de un elemento y qué     #
# community se quiere, determinar cuál es la community a emplear.    #
# Realiza una búsqueda en la base de datos. Si el elemento tiene     #
# aquí especificada la community, esta es empleada. En caso          #
# contario, extrae la community de las opciones generales            #
# especificadas en config.xml                                        #
######################################################################
def determinarComm(self, ip, rorw):

    buscaComm = QSqlQuery()
    consulta = "SELECT " + rorw + " FROM elementos WHERE ip_address='" + ip + "';"
    buscaComm.exec_(consulta)
    size = 0
    while buscaComm.next():
        size += 1
        if size > 1:
            QMessageBox.warning(None, "Error", "Tiene usted un problema de IP duplicada. Por favor, solucióne lo.")
            return
        else:
            try:
                if buscaComm.value(0).isNull:
                    tree = ET.parse('config.xml')
                    for item in tree.iter():
                        if item.tag == rorw:
                            comm = item.text
            except:
                comm = buscaComm.value(0)
    return comm
```

Figura 5.5: **Desarrollo del caso de uso CambiarValores *determinarComm*.**

```
if communityro == '' or communityrw == '' or communityro == None or communityrw == None:
    QMessageBox.warning(None, "Error de configuración", "Ha ocurrido un error en la lectura de la configuración.")
    return
```

Figura 5.6: **Desarrollo del caso de uso CambiarValores. Comprobación de *community*.**

```
elif consulta == "set":
    self.miServidor.snmpset(argumento, direccion, communityrw, nuevoValor[0])
```

Figura 5.7: **Desarrollo del caso de uso CambiarValores. Llamada al servidor.**

```
#####################################################################
#                       Método: snmpset                             #
#                                                                   #
# El cometido de este método es alterar el valor de un OID de un    #
# agente determinado. Toma como parámetros de entrada el OID objeto #
# de la operación (argumento), la dirección IP del elemento         #
# gestionado (direccion), la community de lectura/escritura         #
# (community) y el nuevo valor que debe ser escrito (nuevoValor).   #
#####################################################################
def snmpset(self, argumento, direccion, community, nuevoValor):
    try:
        errorIndication, errorStatus, errorIndex, varBinds = next(
            setCmd(SnmpEngine(),
                   CommunityData(community),
                   UdpTransportTarget((direccion, 161)),
                   ContextData(),
                   ObjectType(ObjectIdentity(argumento), nuevoValor)))
    except error.SmiError:
        QMessageBox.warning(None, "Error", "Revise el OID introducido, no parece ser correcto.")
        return

    if errorIndication:
        QMessageBox.warning(None, "Error", str(errorIndication))
    elif errorStatus:
        QMessageBox.warning(None, "Advertencia", "El OID " +
                            str(errorIndex and varBinds[int(errorIndex) - 1][0] or '?') +
                            " no admite escritura. Sus cambios no han tenido efecto")
    else:
        QMessageBox.information(None, "¡Éxito!", "Sus cambios han sido correctamente escritos en el agente")
```

Figura 5.8: **Desarrollo del caso de uso CambiarValores.** ***snmpset***.

munity con la que realizar la operación y `nuevoValor` como el valor del cambio.

Emplea las funcionalidades ofrecidas por *PySNMP* para realizar la operación sobre el agente e informa al usuario, en función de la respuesta del agente (o la no respuesta), del éxito o fracaso de su operación, finalizando así con el desarrollo del presente caso de uso. Este método puede ser visto en la figura 5.8.

5.2.2. Desarrollo del caso de uso EnviarAlarma

Si analizamos el caso de uso EnviarAlarma, podemos ver que el primer supuesto que debe darse es el envío de una *trap* por uno de los agentes presentes en el rango de red gestionado. Si queremos recibirla, tendremos que desarrollar un receptor de «trampas» SNMP.

Dicho esto, deberemos abrir un puerto (en concreto, el 162) destinado a recibir estas notificaciones por parte de los agentes. Nótese que, al tratarse de un servicio que debe correr durante todo el tiempo que el programa se encuentre en ejecución, resulta necesario un nuevo proceso que se dedique específicamente a la recepción de estas *traps*, sin interrumpir el flujo normal del programa.

```
# Inicialización
def __init__(self):
    super(SnmpServer, self).__init__()

    # Abrimos un nuevo proceso, compartiendo con él una variable de tipo lista de listas
    # para almacenar las traps. Utilizamos para ello un objeto de la clase Manager, declarando
    # también la estructura de dicha listas, de cara a que puedan ser añadidas nuevas traps
    # a su llegada.
    manager = Manager()
    fila = manager.list([0, "Dos", "Tres"])
    self.listaTrapReceiver = manager.list([fila, fila])
    trapReceiver = Process(target=self.trapReceiver, name="Trap-receiver", daemon=True, args=(self.listaTrapReceiver,))
    try:
        trapReceiver.start()
    except:
        QMessageBox.warning(None, "Error", "No se ha podido abrir el puerto de escucha de traps. "
                                           "Compruebe que dispone de los privilegios necesarios. "
                                           "El programa puede seguir funcionando, pero no recibirá "
                                           "traps.")
```

Figura 5.9: **Desarrollo del caso de uso EnviarAlarma. Creación de un nuevo proceso**.

```
# UDP sobre IPv4, escuchando en *:162
try:
    config.addTransport(snmpEngine, udp.domainName + (1,), udp.UdpTransport().openServerMode(('', 162)))
except error.CarrierError:
    print("No ha podido ser abierto el puerto 162 para la recepción de traps."
          " El programa seguirá funcionando, pero no podrá recibir traps.")
    return
```

Figura 5.10: **Desarrollo del caso de uso EnviarAlarma. Apertura de puerto**.

Esto lo realizamos a la inicialización de cualquier objeto de la clase `SnmpServer`, como podemos observar en la figura 5.9.

Aquí, además, creamos una lista, gestionada por un `Manager`, que nos permitirá la modificación posterior de dicha lista por el proceso creado como `trapReceiver`. Este parámetro será quien almacene las «trampas» llegadas, como una lista de listas (que tendrá elemento afectado, valor del *trap* y fecha y hora de llegada, por cada fila). Debemos inicializarla de esta manera aquí debido a que si pasamos al proceso nuevo como argumento una simple lista, no podremos crear dentro de él una lista de listas. Estas dos primeras filas, introducidas de manera previa a la creación del proceso, no serán tenidas en cuenta a la hora de la representación de las alarmas en la ventana principal.

El nuevo proceso, desarrollado en el método `trapReceiver`, trata de abrir el puerto 162, escuchando para todas las direcciones (`*:162`). En caso de que no consiga su cometido, se informará al usuario de que el programa seguirá su curso, pero no podrá recibir *traps*. Esto podría ocurrir, por ejemplo, ante una falta de privilegios. En la figura 5.10 podemos observar cómo se realiza esta apertura y gestión de la posibilidad de que no pudiera ser abierto el puerto en cuestión.

```
# Contexto de ejecución
execContext = snmpEngine.observer.getExecutionContext('rfc3412.receiveMessage:request')

# Obtenemos la dirección IP del agente que envía la notificación
agente = str(execContext['transportAddress'][0])

# Recorremos la notificación y la guardamos. Además, generamos una marca temporal
# y la guardamos también con el resto de los campos de la trampa
# (agente, trap, hora). Así se representará en la ventana principal
for name, val in varBinds:
    trampa = []
    trapPretty = str(name.prettyPrint()) + " = " + str(val.prettyPrint())
    trampa = [agente, trapPretty]
    trampa.append(str(datetime.datetime.today()))
    listaRecibida.append(trampa)
```

Figura 5.11: **Desarrollo del caso de uso EnviarAlarma. Recepción de *traps*.**

Ante la llegada de una notificación, y gracias a las funcionalidades ofrecidas por *PySNMP*, dicha «trampa» es recogida y añadida a la lista (que ha sido llamada en este método como `listaRecibida`) junto con la dirección IP del elemento que la envía y una marca horaria. Esta gestión podemos verla en la figura 5.11.

Por la parte de la vista, cada 10 segundos, se revisará la información contenida en esta lista y se dibujará la tabla de alarmas en función de ella, finalizando con el desarrollo del presente caso de uso, como podemos ver en la figura 5.12.

5.2.3. Desarrollo del caso de uso BuscarElemento

Nos encontramos ahora ante el desarrollo del caso de uso «BuscarElemento». Deberemos escribir el código necesario para mostrar al usuario un diálogo expandible en el que poder introducir los parámetros para una búsqueda en la base de datos y la posterior representación de resultados. Es por ello que podríamos dividir este desarrollo en tres partes. El desarrollo del diálogo expandible, la consulta a la base de datos y la representación de resultados.

```
########################################################################
#                  Método: updateTablaAlarmas2                         #
#                                                                      #
# El presente método se encargará de dibujar la tabla de alarmas en #
# función del contenido de la variable listaTrapReceiver, de         #
# SnmpServer. Será llamado periódicamente a fin de mantener esta     #
# tabla actualizada.                                                   #
# Inicialmente, y debido a las limitaciones de ser una managerlist, #
# se recorre entera para determinar su tamaño (no se dispone del     #
# método len). De ahí se eliminan las 2 primeras líneas (preámbulo) #
# y se dibuja el resto del contenido de la lista de listas, que      #
# alberga las traps recibidas durante la ejecución.                    #
# Inicialmente se lee el fichero a fin de determinar el número de    #
########################################################################
def updateTablaAlarmas2(self):

    numLineas = 0
    for linea in self.miControlador.miServidor.listaTrapReceiver:
        numLineas += 1

    self.tablaAlarmas.setRowCount(numLineas-2)

    i = 0
    for linea in self.miControlador.miServidor.listaTrapReceiver:
        if linea[0] is 0:
            pass

        else:
            elementoAfectado = QTableWidgetItem(linea[0])
            valor = QTableWidgetItem(linea[1])
            llegada = QTableWidgetItem(linea[2])

            self.tablaAlarmas.setItem(i, 0, elementoAfectado)
            self.tablaAlarmas.setItem(i, 1, valor)
            self.tablaAlarmas.setItem(i, 2, llegada)

            i += 1

    self.tablaAlarmas.resizeColumnsToContents()
```

Figura 5.12: **Desarrollo del caso de uso EnviarAlarma. Representación.**

Desarrollo del diálogo expansible

La primera, destinada a la visualización inicial del diálogo expandible, a fin de introducir hasta 8 opciones de búsqueda, como se enuncia en el caso de uso requiere de un elemento de *PyQt* llamado `QFrame`. Este *widget* permite que parte de un *layout* pueda quedar oculto durante cierto momento de la ejecución del programa.

Así, especificaremos 8 campos de la dupla `QComboBox`/`QLineEdit`, dejando mostrado uno de ellos y ocultos los 7 restantes. La `QComboBox` estará formada por las opciones que se guardarán en la base de datos, a saber,

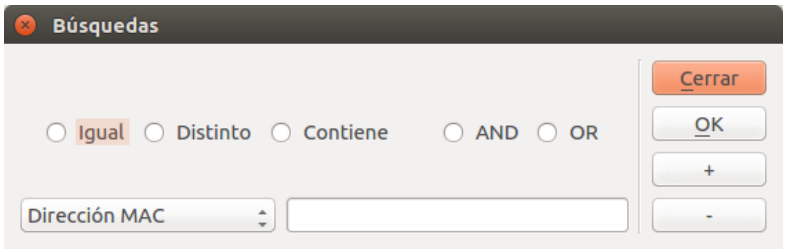

Figura 5.13: **Desarrollo del caso de uso BuscarElemento. Diálogo inicial comprimido**.

dirección MAC, dirección IP, último contacto, *community* de sólo lectura y *community* de lectura/escritura. La `QLineEdit` servirá para que el usuario introduzca el valor de búsqueda asociado a la selección realizada en la `QComboBox`.

Además de esta parte del diálogo, inmediatamente superior a ella, se encuentran dos grupos de radiobotones dispuestos horizontalmente. El primer grupo está destinado a elegir si se quiere que la búsqueda se realice para que el parámetro introducido por el usuario sea igual, distinto o parte del contenido en la base de datos. El segundo grupo determinará si la búsqueda se realizará como una AND o como una OR lógica, para el caso de varios parámetros.

Paralelamente a todo esto, se mostrará una botonera, con 4 botones dispuestos en vertical, con el orden «Cerrar», «Ok», «+» y «-». El botón «Cerrar», como su propio nombre indica, cerrará el diálogo, mientras que el botón «Ok» lanzará la consulta y los botones «+» y «-» servirán para añadir y eliminar campos a la consulta que deberá tener al menos 1 y no más de 8.

De esta manera, en la figura 5.13 podemos ver el diálogo mostrado inicialmente, mientras que en la figura 5.14 se observa el diálogo completamente expandido.

La composición final es una `QHBoxLayout`, agrupando tres elementos. Comenzando por la izquierda, aquel que comprende los radiobotones y las duplas `QComboBox` / `QLineEdit`, una línea vertical de separación y la botonera.

El primer elemento mencionado es una `QVBoxLayout`, con nueve elementos, dispuestos de forma vertical. El primero de ellos es una `QHBoxLayout` con los dos grupos de radiobotones (`QGroupBox`), el segundo es otra `QHBoxLayout` con la dupla `QComboBox`/`QLineEdit` y el resto, `QFrameLayouts`. Recordemos que esto es así para que puedan ser «escondidos», ya que dentro de la

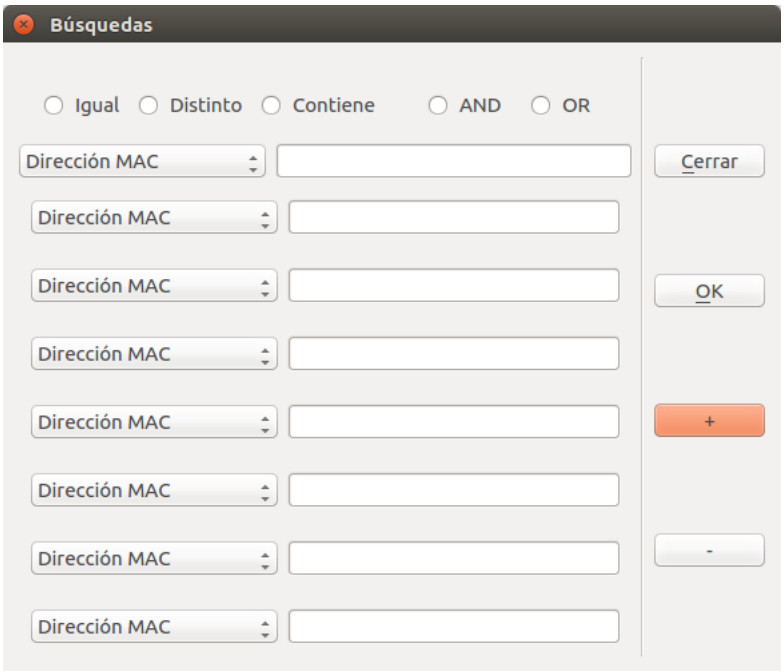

Figura 5.14: **Desarrollo del caso de uso BuscarElemento. Diálogo inicial expandido**.

```
# "Escondemos" los 7 campos restantes, que serán mostrados si el usuario así lo quiere
self.moreFrame.hide()
self.moreFrame2.hide()
self.moreFrame3.hide()
self.moreFrame4.hide()
self.moreFrame5.hide()
self.moreFrame6.hide()
self.moreFrame7.hide()
```

Figura 5.15: **Desarrollo del caso de uso BuscarElemento. Campos inicialmente ocultos**.

`QFrameLayout` encontramos una `QHBoxLayout` idéntica a la que es mostrada inicialmente.

En la figura 5.15 puede verse cómo se consigue que estos 7 campos estén inicialmente ocultos. No vamos a mostrar el código completo por considerarlo de poco interés, repetitivo y suficientemente explicado, pero puede consultarse, si así se desea, en el apéndice B (`busquedas.py`).

Como comentamos anteriormente, la pulsación en los botones «+» y «-» añadía y eliminaba campos de búsqueda, siendo el mínimo admisible 1 y el máximo 8. Hemos desarrollado dos métodos, conectados a la pulsación de estos botones, que consiguen el cometido deseado. Estos son `addField()` y

```
#####################################################################
#                      Método: addField                             #
#                                                                   #
# Este método es lanzado cuando el usuario pulsa sobre el botón "+" #
# del presente diálogo. Se apoya en el parámetro moreFramesVisible, #
# al que actualiza si fuera el caso, para determinar qué trama debe #
# mostrar y para, llegado al límite, señalar al usuario que no es   #
# posible añadir más parámetros a la búsqueda.                      #
#####################################################################
def addField(self):

    if self.moreFramesVisible == 0:
        self.moreFrame.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 1:
        self.moreFrame2.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 2:
        self.moreFrame3.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 3:
        self.moreFrame4.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 4:
        self.moreFrame5.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 5:
        self.moreFrame6.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 6:
        self.moreFrame7.show()
        self.moreFramesVisible += 1
    else:
        QMessageBox.warning(None, self.tituloAdvertencia, "El máximo de parámetros para la búsqueda es de 8")
```

Figura 5.16: **Desarrollo del caso de uso BuscarElemento. Método *addField*.**

`deleteField()`, y pueden verse en las figuras 5.16 y 5.17 respectivamente.

Para ello, se apoyan en una variable pública de la clase a la que pertenecen, `Buscar`, llamada `moreFramesVisible`, que se encuentra inicialmente con valor entero 1, y esconden o destapan campos en función de ella.

Desarrollo de la consulta a la base de datos

Una vez el usuario ha pulsado «Ok», se comprueba si su consulta cumple con los requisitos para ser procesada (debe haber elegido una opción de cada grupo de radiobotones) y se realizan los cambios oportunos de los valores de estas selecciones a valores lógicos en formato SQL.

Esto se realiza en el método `accept` de la clase `Buscar`, conectado al botón que hemos mencionado previamente.

La comprobación y traducción de valores puede contemplarse en la figura 5.18.

```
######################################################################
#                       Método: deleteField                          #
#                                                                    #
# Este método es lanzado cuando el usuario pulsa sobre el botón "-" #
# del presente diálogo. Se apoya en el parámetro moreFramesVisible, #
# al que actualiza si fuera el caso, para determinar qué trama debe #
# esconder y para, llegado al límite, señalar al usuario que no es  #
# posible realizar una búsqueda sin parámetros.                      #
######################################################################
def deleteField(self):

    if self.moreFramesVisible == 7:
        self.moreFrame7.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 6:
        self.moreFrame6.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 5:
        self.moreFrame5.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 4:
        self.moreFrame4.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 3:
        self.moreFrame3.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 2:
        self.moreFrame2.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 1:
        self.moreFrame.hide()
        self.moreFramesVisible -= 1
    else:
        QMessageBox.warning(None, self.tituloAdvertencia, "No se puede realizar una búsqueda sin parámetros")
```

Figura 5.17: **Desarrollo del caso de uso BuscarElemento. Método *deleteField*.**

```
# Comprobaciones de los radiobotones
if not (self.andLogica.isChecked() or self.orLogica.isChecked()):
    QMessageBox.warning(None, self.tituloAdvertencia, "Debe seleccionar si desea búsqueda como AND o como OR.")
    return
if not (self.igual.isChecked() or self.noIgual.isChecked() or self.contiene.isChecked()):
    QMessageBox.warning(None, self.tituloAdvertencia, "Debe seleccionar el tipo de búsqueda.")
    return

# Extraemos la información de los radiobotones y la convertimos a notación SQL
if self.igual.isChecked():
    simbolo = "="
elif self.contiene.isChecked():
    simbolo = "LIKE"
else:
    simbolo = "!="

if self.andLogica.isChecked():
    logica = "AND"
else:
    logica = "OR"
```

Figura 5.18: **Desarrollo del caso de uso BuscarElemento. Traducción de valores lógicos.**

```
#####################################################################
#                     Método: buscarValores                         #
#                                                                   #
# El presente método prepara la consulta a la base de datos.        #
# Tomando como entradas el símbolo y la lógica en notación MySQL,   #
# recorre cada campo de inserción de la lista de QLineEdit y, si el #
# valor en ella no es nulo, relaciona éste con el valor del         #
# desplegable y lo asocia a la columna adecuada de la tabla         #
# "elementos" de la base de datos.                                  #
#####################################################################
def buscarValores(self, simbolo, logica):

    # Inicializaciones. "i" servirá para ir repasando las posiciones de la lista de QLineEdit,
    # "campos" nos servirá para introducir el operador lógico en el caso de que el usuario haya
    # seleccionado más de 1 campo y la lista "índices" nos relacionará el valor que podemos
    # obtener de las comboBoxes con la columna correspondiente de la tabla "elementos".
    i = 0
    campos = 0
    indices = ["mac_address", "ip_address", "lastTimeDetected", "communityRO", "communityRW"]

    # Realizamos el bucle y preparamos la consulta
    for caja in self.comboBoxes:
        if self.inserciones[i].text() is not '':
            if campos != 0:
                self.preparatoria = self.preparatoria + logica + " "
            if simbolo != "LIKE":
                self.preparatoria = self.preparatoria + indices[caja.currentIndex()] + " " + simbolo + " '" + \
                    self.inserciones[i].text() + "' "
            elif simbolo == "LIKE":
                self.preparatoria = self.preparatoria + indices[caja.currentIndex()] + " " + simbolo + " '%" + \
                    self.inserciones[i].text() + "%' "

            campos += 1
        i += 1

    # Le introducimos a la consulta, una vez finalizado el bucle, el ';' final necesario.
    self.preparatoria = self.preparatoria + ";"
```

Figura 5.19: **Desarrollo del caso de uso BuscarElemento. Método *buscarValores*.**

Posteriormente, se llama al método `buscarValores`, con el símbolo y lógica en formato SQL. Este método recorre las distintas duplas de `QComboBox` / `QLineEdit` a fin de extraer los valores y anexarlos a una cadena de texto en la que escribimos la consulta, almacenada con el nombre de `preparatoria`, inicialmente con el valor `"SELECT * FROM elementos WHERE "`. Este método puede estudiarse en la figura 5.19.

Una vez se ha actualizado la consulta con todos los parámetros, nos queda ejecutarla, cosa que hacemos desde el método `accept`, pasando posteriormente el resultado de dicha búsqueda al nuevo diálogo que representará los resultados (`RepresentarResultadoBuscar`) y volviendo a dejar la variable pública `preparatoria` con su valor inicial, esperando nuevas consultas, como podemos ver en la figura 5.20.

Representación del resultado de la búsqueda

Representaremos los resultados de la búsqueda en forma de tabla, con 5 columnas, que serán las existentes en la tabla `elementos` de nuestra base de

```
# Ejecutamos la búsqueda con la variable "preparatoria" lista como petición SQL gracias
# al método buscarValores
self.busqueda.exec_(self.preparatoria)

# Creamos una instancia de la clase RepresentarResultadoBuscar, que mostrará el resultado en forma
# de tabla en una ventana emergente, pasándole la petición realizada, a fin de que se extraigan
# los resultados de esta y se representen de un modo "user-friendly".
miPrueba = RepresentarResultadoBuscar(self.busqueda)
miPrueba.exec_()

# Una vez ejecutada la consulta, dejamos lista la variable preparatoria para futuras búsquedas.
self.preparatoria = "SELECT * FROM elementos WHERE "
```

Figura 5.20: **Desarrollo del caso de uso BuscarElemento. Ejecución de la consulta**.

```
    # Obtenemos el número de filas de la respuesta dada por la BBDD y fijamos
    # con ello el número de filas de la QTableWidget.
    numFilas = 0
    while query.next():
        numFilas += 1
    self.tablaResultados.setRowCount(numFilas)
```

Figura 5.21: **Desarrollo del caso de uso BuscarElemento. Número de filas**.

datos.

No nos detendremos en mostrar el dibujo de esta tabla, sino que expondremos cómo se recorre la respuesta de la base de datos a fin de representarla.

Inicialmente, determinaremos el número de resultados arrojados por nuestra consulta, recorriendo la respuesta fila a fila, como puede verse en la figura 5.21, a fin de determinar el número de filas de la tabla a representar.

Tras esto, nos posicionamos al inicio de la respuesta y la recorremos de nuevo, extrayendo los valores e insertándolos en los campos de la tabla, como podemos ver en la figura 5.22, finalizando así con el desarrollo del caso de uso que nos ocupa.

5.2.4. Desarrollo del caso de uso ResincronizarTablaElementos

Para comenzar con el flujo normal de este caso de uso, el usuario deberá hacer *click* sobre el icono llamado a resincronizar en la barra de herramientas de la parte superior de la ventana principal. Una vez recogida esta interacción, la ventana principal traslada el control del proceso al controlador central.

```
# Como estábamos al final de la query tras el bucle que determina el número de filas,
# nos colocamos al principio para el bucle que rellena esas filas.
# Posteriormente recorremos el resultado de la búsqueda para rellenar las celdas de la GUI.
# Como podemos encontrarnos ante valores nulos en los campos de las Communities, debemos
# manejar las posibles excepciones que esto pueda arrojarnos.
query.seek(-1)
i = 0
while query.next():
    mac_address = QTableWidgetItem(query.value(0))
    ip_address = QTableWidgetItem(query.value(1))
    lastTimeDetected = QTableWidgetItem(query.value(2))

    try:
        if query.value(3).isNull:
            communityRO = QTableWidgetItem('')
    except:
        communityRO = QTableWidgetItem(query.value(3))

    try:
        if query.value(4).isNull:
            communityRW = QTableWidgetItem('')
    except:
        communityRW = QTableWidgetItem(query.value(4))

    self.tablaResultados.setItem(i, 0, mac_address)
    self.tablaResultados.setItem(i, 1, ip_address)
    self.tablaResultados.setItem(i, 2, lastTimeDetected)
    self.tablaResultados.setItem(i, 3, communityRO)
    self.tablaResultados.setItem(i, 4, communityRW)

    i += 1

self.tablaResultados.resizeColumnsToContents()
self.tablaResultados.resizeRowsToContents()
self.muestra = QVBoxLayout()
self.muestra.addWidget(self.tablaResultados)

self.setLayout(self.muestra)
```

Figura 5.22: **Desarrollo del caso de uso BuscarElemento. Representación del resultado**.

Es entonces cuando entra en juego el fichero de configuración. La resincronización se realizará para todo el rango definido en dicho fichero. Para ello, el primer campo, como pudimos ver en la figura 4.5, es llamado `Subnet` e indica la dirección de subred sobre la que realizar el descubrimiento. Es complementada con el siguiente campo (`Prefix`), que indica el número de bits a «1» de la máscara de subred. Además, al tratarse de un descubrimiento, se tomará la *community* por defecto, pues podríamos encontrarnos ante elementos que no tuviésemos almacenados en la base de datos. Como las operaciones relacionadas con este caso de uso serán *get* y *walk*, la *community* que tomaremos del fichero de configuración será la de sólo lectura, es decir, el tercer campo, *CommunityRO*. Podemos ver, en la figura 5.23, cómo se extraen estos parámetros del fichero de configuración, haciendo uso de la biblioteca `xml.etree.cElementTree` o, si esta no estuviera disponible, de `xml.etree.ElementTree`.

```
# Elegimos nuestro fichero de configuración
tree = ET.parse('config.xml')

# Extraemos de dicho fichero dirección, la máscara de subred y la community de lectura
for elem in tree.iter():
    if elem.tag == 'Subnet':
        subnetIpAddress = elem.text
    elif elem.tag == 'Prefix':
        self.subnetPrefix = int(elem.text)
    elif elem.tag == 'CommunityRO':
        communityRO = elem.text
```

Figura 5.23: **Desarrollo del caso de uso ResincronizarTablaElementos. Lectura del fichero de configuración**.

Una vez obtenidos estos parámetros, el controlador central calculará el primer y el último elemento del rango. Para ello, transformará el prefijo en una máscara de subred (por ejemplo, un prefijo 25 pasaría a ser una máscara de subred 255.255.255.128) y efectuará, *byte* a *byte*, una AND *bitwise* lógica de manera que, si la dirección introducida en el fichero de configuración no correspondiese con una dirección de subred adecuada, se convertiría así. Un ejemplo puede ser un campo `Subnet` con valor 192.168.1.7 y campo `Prefix` 25. Nuestro método determinaría que la dirección de subred es 192.168.1.0/25, con primer elemento a inspeccionar 192.168.1.1 y último 192.168.1.126, pues eliminaría del rango a inspeccionar la dirección de *broadcast* y la de subred. El algoritmo empleado para determinar este rango puede consultarse en el apéndice C (`controlador.py`).

Tras determinar el rango, el controlador central actuará junto al servidor SNMP a fin de realizar un *snmpget* en que solicitará el *sysObjectID* a cada elemento presente en el rango calculado. Si existe respuesta, en esa dirección existe un elemento y se almacena temporalmente en un diccionario llamado `elementos`, con clave su dirección IP y valor el *sysObjectID* concreto del elemento.

A fin de no sobrecargar el equipo en que se ejecuta el programa ni saturar la red con tráfico innecesario, se pedirá al usuario confirmación para realizar el descubrimiento ante redes de 510 o más posibles elementos, como podemos ver en las figuras 5.24, 5.25 y 5.26. En caso de que el usuario confirme que desea ejecutar la operación, se seguirá adelante y se realizará el escaneo. En caso contrario, se abortará la acción.

En la figura 5.27 mostramos el algoritmo desarrollado para el caso de máscaras de 24 o más bits, en el que, como hemos mencionado, realizamos un *snmpget* del *sysObjectID* y, si hay respuesta, almacenamos el posible ele-

```
# Máscara entre 0 y 7 bits
if primerByte:
    confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar una resincronización para"
                                                      " 33.554.430 o más direcciones. ¿Seguro que quiere continuar?"
                                                      " Puede modificar la configuración si lo desea antes de"
                                                      " realizar esta resincronización. Pulsar 'OK' continuaría"
                                                      " con la resincronización.", QMessageBox.Ok, QMessageBox.Cancel)
```

Figura 5.24: **Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (1).**

```
# Máscara entre 8 y 15 bits
elif segundoByte:
    confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar una resincronización para"
                                                      " 131.070 o más direcciones. ¿Seguro que quiere continuar?"
                                                      " Puede modificar la configuración si lo desea antes de"
                                                      " realizar esta resincronización. Pulsar 'OK' continuaría"
                                                      " con la resincronización.", QMessageBox.Ok, QMessageBox.Cancel)
```

Figura 5.25: **Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (2).**

```
# Máscara entre 16 y 23 bits
elif tercerByte:
    confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar una resincronización para"
                                                      " 510 o más direcciones. ¿Seguro que quiere continuar?"
                                                      " Puede modificar la configuración si lo desea antes de"
                                                      " realizar esta resincronización. Pulsar 'OK' continuaría"
                                                      " con la resincronización.", QMessageBox.Ok, QMessageBox.Cancel)
```

Figura 5.26: **Desarrollo del caso de uso ResincronizarTablaElementos. Confirmación (3).**

```
# Máscara de 24 o más bits
else:
    i = primero[3]
    while i <= ultimo[3]:
        posibleElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(primero[2]) + '.' + str(i)
        print("Inspeccionando la dirección " + posibleElemento)
        respuestaDiscover = self.miServidor.snmpget(".1.3.6.1.2.1.1.2.0", posibleElemento, communityRO)
        i = i+1
        if respuestaDiscover is not None:
            respuestaDiscover = str(respuestaDiscover)
            ipElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(primero[2]) + '.' + str(i-1)
            elementos[ipElemento] = respuestaDiscover
```

Figura 5.27: **Desarrollo del caso de uso ResincronizarTablaElementos. Algoritmo de descubrimiento**.

```
######################################################################
#                       Método: snmpget                              #
#                                                                    #
# Se encarga de realizar un get, tomando como argumentos el OID      #
# objetivo (consulta), la dirección del agente objetivo (direccion)  #
# y la community, que en este caso debería ser de sólo lectura.      #
######################################################################
def snmpget(self, consulta, direccion, community):
    try:
        g = getCmd(SnmpEngine(), CommunityData(community), UdpTransportTarget((direccion, 161)),
               ContextData(),ObjectType(ObjectIdentity(consulta)))
        errorIndication, errorStatus, errorIndex, varBinds = next(g)
    except:
        return

    if errorIndication:
        print(errorIndication)
    elif errorStatus:
        print('%s at %s' % (errorStatus.prettyPrint(), errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
    else:
        for varBind in varBinds:
            return varBind
```

Figura 5.28: **Desarrollo del caso de uso ResincronizarTablaElementos. Método *snmpget***.

mento.

Para que la clase `SnmpServer` realice el *get* se ha desarrollado un método gracias a las facilidades ofrecidas por *PySNMP* denominado `snmpget`, que puede verse en la figura 5.28.

Una vez rellenado el diccionario con los posibles elementos, los introduciremos en la base de datos. Los valores de nuestra petición serán la dirección MAC, la dirección IP y una marca temporal, generada en el momento del envío a la base de datos.

Nos encontramos ante dos posibles escenarios para la consulta, ya que el elemento puede o no formar parte de la base de datos previamente a realizar la resincronización. Es por ello que, inicialmente, preparamos una consulta de actualización con la dirección MAC como elemento a comparar. Si no existen filas afectadas por dicha petición, efectuamos otra, esta vez de inserción.

```
# Creamos una instancia QSqlQuery para realizar nuestra petición
petiSql = QSqlQuery()

# Preparamos una actualización si la MAC del elemento ya estaba en la BBDD.
# Podría darse un error de MAC duplicada que no captáramos, pero, dado que su aparición sucede
# con poca frecuencia, lo dejamos fuera del alcance de nuestra aplicación, siendo el usuario el que
# deberá proporcionar los adecuados métodos para evitar este problema.
peticion1 = "UPDATE elementos SET ip_address='" + diccionarioQuery["ip_address"] +\
            "', lastTimeDetected='" + str(diccionarioQuery["lastTimeDetected"]) +\
            "' WHERE mac_address='"+ diccionarioQuery["mac_address"] + "';"

# Ejecutamos la petición previamente preparada
petiSql.exec_(peticion1)

# Si no existía ningún elemento con esa MAC, no habremos escrito nada en la BBDD y, por lo tanto,
# preparamos la consulta de inserción con los datos presentes en nuestro diccionario.
if petiSql.numRowsAffected() == 0:
    peticion2 = "INSERT INTO elementos (mac_address, ip_address, lastTimeDetected) VALUES ('"\
                + diccionarioQuery["mac_address"] + "', '" + \
    diccionarioQuery["ip_address"] + "', '" + str(diccionarioQuery["lastTimeDetected"]) + "');"

    # Ejecutamos la petición previamente preparada
    try:
        petiSql.exec_(peticion2)
    except:
        QMessageBox.warning(None, "Error SQL", petiSql.lastError().text())
```

Figura 5.29: **Desarrollo del caso de uso ResincronizarTablaElementos. Actualización de la base de datos**.

Podemos ver este procedimiento en la figura 5.29.

Con esto finalizaría el método `resync` de la clase `ControladorCentral`. Ahora bien, el lector podría pensar, acertadamente, que no se han obtenido todos los parámetros que son mostrados en la ventana principal. Es cierto, eso es tarea de otro método. Esto es así porque, al no almacenar más que los 5 campos descritos en varias secciones previas para la base de datos, debemos realizar un *walk* del módulo *SNMPv2-MIB::system* como parte de la secuencia de arranque, a fin de recabar *sysDescr*, *sysName*, *sysContact*, *sysObjectID*, *sysUpTime*, *sysLocation* y *sysServices*, que son mostrados en la pantalla principal. No vamos a dejar de lado esto ahora que no estamos ante una secuencia de arranque, pero vamos a compartir el método.

Tras la finalización de `resync` y el retorno del proceso a la ventana principal, se llama a otro método de esta clase, `updateTablaElementos`, que tiene como fin rellenar, precisamente, la tabla de elementos, con valores actualizados. Para ello, efectúa una lectura de la base de datos y solicita al controlador la realización de los *walk* necesarios.

El método llamado en el controlador es `wakeUp(ipElemento)` y, básicamente, se encarga de solicitar al servidor SNMP un *snmpwalk* del módulo mencionado, extrayendo de su respuesta los parámetros *sysObjectID*, *sysDescr*, *sysUpTime*, *sysContact*, *sysName*, *sysLocation* y *sysServices*, que son

```
def wakeUp(self, ipAddr):

    comm = self.determinarComm(ipAddr, 'communityRO')

    try:
        valores, etiquetas = self.miServidor.snmpwalk(ipAddr, comm, False, 'SNMPv2-MIB')

        i = 0
        for elem in valores:
            if i == 0:
                sysDescr = elem
            elif i == 1:
                sysObjectID = elem
            elif i == 2:
                sysUpTime = elem
            elif i == 3:
                sysContact = elem
            elif i == 4:
                sysName = elem
            elif i == 5:
                sysLocation = elem
            elif i == 6:
                sysServices = elem
            i += 1
        lastTimeDetected = datetime.datetime.today()
    except:
        return

    return str(lastTimeDetected), sysDescr, sysObjectID, sysUpTime, sysContact, sysName, sysLocation, sysServices
```

Figura 5.30: **Desarrollo del caso de uso ResincronizarTablaElementos. Método *wakeUp***

devueltos para su representación. También devuelve una marca temporal a fin de que el último contacto sea verídico (esto no tiene demasiada importancia para nuestro caso porque lo acabaríamos de escribir en la base de datos, pero en la secuencia de arranque toma especial relevancia). El método `wakeUp` podemos verlo en la figura 5.30.

Por su parte, el método `snmpwalk` de la clase `SnmpServer`, desarrollado gracias a las facilidades ofrecidas por *PySNMP*, será explicado con mayor detalle en la subsección 5.2.6.

Una vez recibida la respuesta del `wakeUp`, procedemos a dibujar los cambios en la tabla mostrada al usuario en la ventana principal. Además, se actualiza en la base de datos la fecha y hora de último contacto, en virtud de la respuesta dada por el controlador central. El código de esta sección puede verse en el apéndice D (`ventanaPrincipal.pyw`).

5.2.5. Desarrollo del caso de uso AplicarConfiguración

Las situaciones en que se aplica la configuración en nuestro proyecto son diversos. Por ejemplo, podríamos hablar del momento en que se determina la *community* a emplear en función de la configuración, como vimos en el caso

```
self.mibCompiler = MibCompiler(SmiV2Parser(), PySnmpCodeGen(), PyFileWriter('/tmp/pysnmp/mibs'))

# Para resoluciones
self.mibBuilder = smi.builder.MibBuilder()
self.mibViewController = smi.view.MibViewController(self.mibBuilder)

# Leemos, de config.xml, los módulos MIB a cargar inicialmente, y llamamos a loadMib para que los cargue
tree = ET.parse('config.xml')
mibUpload = []
for elem in tree.iter(tag='MibUpload'):
    mibUpload.append(elem.text)

self.loadMib(*mibUpload)
```

Figura 5.31: **Desarrollo del caso de uso AplicarConfiguración**.

de uso CambiarValores o, también, podríamos averiguar cuál es el rango de red según lo existente en el fichero de configuración. Pero, si hay algo que aún no hemos abordado relacionado con este caso de uso es la carga inicial de módulos MIB en función del contenido del fichero de configuración.

Esto se realiza en la secuencia de arranque. Una instancia de la clase `MainWindow` es llamada desde el `main()`. Esta clase, en su inicialización, instancia un objeto de la clase `ControladorCentral`, que, a su vez, instancia un objeto de la clase `SnmpServer` en su inicialización.

¿Y qué tiene esta clase relacionado con el caso de uso que nos ocupa? Además de crear el proceso del `trapReceiver` visto en el caso de uso EnviarAlarma, crea un compilador MIB, lee el fichero de configuración y solicita la carga de todos los módulos existentes en dicho fichero a través de su método `loadMib`, que será explicado en profundidad cuando hablemos del desarrollo del caso de uso CargarMIB.

El cómo se realiza esta extracción de la configuración y se llama al método mencionado unas líneas más arriba, podemos verlo en la figura 5.31.

5.2.6. Desarrollo del caso de uso GenerarInforme

El caso de uso que nos ocupa toma, en la secuencia deseada, uno de los pasos previos que ya hemos visto en el caso de uso CambiarValores. Este es el diálogo presentado para elegir cambiar algún valor, eliminar el elemento, modificar sus *communities* específicas, realizar un *walk* completo, o bien, realizar un *walk* de un subárbol MIB.

Nos centraremos, por lo tanto, en estas últimas dos opciones, por considerarlas informes del elemento seleccionado, bien al completo, bien de un

```
if miDlgEleccion.getNextOrdenado:
    valores, etiquetas = self.miControlador.peticiones("getnext", miDlgEleccion.insercionModulo.text(), ipElemento)
    miArbol = dialogosAdicionales.DlgArbol(valores, etiquetas)
    miArbol.exec_()

if miDlgEleccion.walkOrdenado:
    valores, etiquetas = self.miControlador.peticiones("walk", "", ipElemento)
    miArbol = dialogosAdicionales.DlgArbol(valores, etiquetas)
    miArbol.exec_()
```

Figura 5.32: **Desarrollo del caso de uso GenerarInforme. Manejo de peticiones**.

```
elif consulta == "getnext":
    try:
        valores, etiquetas = self.miServidor.snmpwalk(direccion, communityro, False, argumento)
    except smi.error.MibNotFoundError:
        QMessageBox.warning(None, "Error", "Su petición no ha podido ser atendida. El módulo"
                                           " que ha introducido no se reconoce como un módulo válido."
                                           " Puede revisar su inserción o bien cargar el módulo"
                                           " que desee.")
        return
    return valores, etiquetas

elif consulta == "walk":
    try:
        valores, etiquetas = self.miServidor.snmpwalk(direccion, communityro, True)
    except TypeError:
        QMessageBox.warning(None, "Error", "Su petición no ha podido ser atendida. Es posible"
                                           "que no tenga comunicación con el agente seleccionado.")
        return
    return valores, etiquetas
```

Figura 5.33: **Desarrollo del caso de uso GenerarInforme. Método *peticiones***.

módulo en concreto.

Como en los casos anteriores, se accederá al diálogo haciendo doble *click* sobre el elemento, mostrado en el *widget* central de la pantalla principal.

En la clase `MainWindow` y en el método `elementoSeleccionado` también se recogerán estas dos peticiones haciendo uso de variables de control análogas a las explicadas para el caso de uso CambiarValores, como podemos comprobar en la figura 5.32, contando como `getnext` el *walk* de un módulo en concreto y como `walk` el del árbol MIB completo de dicho elemento.

El método `peticiones` de `ControladorCentral` cogerá el testigo, realizando las peticiones oportunas al servidor SNMP y almacenando los valores y las etiquetas devueltas. Esto es así para poder posteriormente dibujar el árbol con las etiquetas en lugar de en formato numérico, con los nodos hoja mostrando el valor adecuado.

Como podemos ver en la figura 5.33, ambas peticiones dirigen al mismo método: `snmpwalk`. Este método se encargará de realizar un *walk* sobre un

subárbol o sobre el árbol MIB completo. Recibe como entradas `direccion`, que será la dirección IP en la que se encuentra el elemento objeto de la petición; `community`, que será la *community* de sólo lectura del agente a inspeccionar; *lexic*, un valor *booleano* que determinará si se realiza la operación sobre un subárbol o sobre el árbol completo, siendo `True` la opción para realizar el *walk* sobre el árbol completo y `False` la que determinará que el *walk* únicamente ha de ser sobre un módulo en concreto; y, por último, `*argumento`. Este posible parámetro determinará el módulo MIB sobre el que realizar el *walk*.

Al realizar la operación (apoyándonos en las facilidades ofrecidas por *PySNMP*) almacenamos en el parámetro `labels` una lista de listas con la identificación de cada OID, obteniendo un formato como el que ese puede ver en el comentario de la figura 5.34. Hemos de añadir que gracias a esto cumplimos también el caso de uso DecodificarOID, al obtener, de un OID, su representación en formato más «amigable» para el ser humano. Por otro lado, los valores de los nodos hoja se van almacenando en una lista, llamada `valores`. De este modo, se tiene una relación entre la posición de las etiquetas de un OID y su valor, que será empleada a fin de representar adecuadamente el informe del elemento en forma de árbol. Así, las etiquetas para un objeto almacenado en `valores[i]` se encontrarían en `etiquetas[i]`, que sería a su vez una lista de N elementos, con N-1 el número de nodos jerárquicamente superiores al nodo hoja.

Una vez obtenido el resultado de la operación, toca representarlo. Para ello, se abrirá un nuevo diálogo, gracias a la clase `DlgArbol`, encargada de representar el árbol obtenido como resultado de nuestra petición inicial.

El método encargado de dicha tarea será `populateTree`. Tomará como entradas los valores y la lista de etiquetas (conocida ahora como `listaSnmp`) y recorrerá cada fila de ésta, representando los elementos en forma de árbol.

Todos los elementos de cada fila que no sean el último, son, seguro, un elemento padre. Es por ello que se obtiene la longitud de cada fila y, para cada elemento que no sea el último (y por tanto, con nodos hijos), se le asignará un valor vacío. El último elemento de cada fila será a quien haya que asignarle el valor almacenado en `valores[k]`, con `k` el número de fila que estemos recorriendo. Para la primera fila, se asignará el primer elemento como `top` y el resto le serán añadidos recursivamente como hijos. Además, todos los elementos serán añadidos como cadenas de texto a la lista `listaHijosStr`. Para el resto de filas se sigue el mismo criterio con los valores, determinando

```
def snmpwalk(self, direccion, community, lexic, *argumento):

    # Almacenaremos en 'labels' una lista de listas con la identificación de cada OID.
    # Por ejemplo, para sysDescr, obtendríamos:
    # ('iso', 'org', 'dod', 'internet', 'mgmt', 'mib-2', 'system', 'sysDescr').
    # Esto es logrado gracias a getLabel() y será usado para la representación en forma de árbol.
    # En la lista valores almacenaremos los valores de estos elementos en un formato legible
    # para el ser humano, gracias a prettyPrint().
    labels = []
    valores = []

    if not argumento:
        argumento = ('SNMPv2-MIB','')

    for (errorIndication,
         errorStatus,
         errorIndex,
         varBinds) in nextCmd(SnmpEngine(), CommunityData(community), UdpTransportTarget((direccion, 161)),
                              ContextData(), ObjectType(ObjectIdentity(argumento[0])), lexicographicMode=lexic):
        if errorIndication:
            print(errorIndication)
            break
        elif errorStatus:
            print('%s at %s' % (errorStatus.prettyPrint(), errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
            QMessageBox.warning(None, "Error", "Ha ocurrido algún problema al realizar el walk. Revise"
                                               " su operación o el estado del agente.")
            break
        else:
            for name, val in varBinds:
                name.resolveWithMib(self.mibViewController)
                labels.append(name.getLabel())
                valores.append(val.prettyPrint())
    return valores, labels
```

Figura 5.34: **Desarrollo del caso de uso GenerarInforme. Método *snmpwalk*.**

si ha de pintarse o no la etiqueta y su lugar en función de si existe o no esta etiqueta en `listaHijosStr`, y gracias al control de `j`, que nos indica en qué posición nos encontramos. En las figuras 5.35 y 5.36 se muestra el código de este algoritmo, que acabaría con el desarrollo del caso de uso que nos ocupa.

5.2.7. Desarrollo del caso de uso ModificarConfiguración

El desencadenante de este caso de uso se encuentra en la pulsación de ratón sobre uno de los iconos presentes en la barra de herramientas. Concretamente, se debe pulsar en la opción «Alterar la configuración», identificada con un engranaje. Con dicho *click* se instancia y ejecuta un diálogo de la clase `EditarConfDlg`.

Este diálogo, en su inicialización, abre el fichero de configuración y llama al método `cargarDialogo`. Este método tiene como fin leer el contenido del fichero de configuración abierto y representarlo. Los primeros campos (`Subnet`, `Prefix`, `CommunityRO` y `CommunityRW`) serán dispuestos como duplas `QLabel`/`QLineEdit`, con el contenido de las cajas de inserción de texto

```
for fila in listaSnmp:
    aux = len(fila)

    # Primera fila
    if i == 0:
        j = 0
        for elem in fila:
            if top is None:
                top = QTreeWidgetItem(self.treeWidget, [str(elem), ""])
                self.listaHijos.append(top)
                self.listaHijosStr.append(elem)

            else:
                if j == (aux -1):
                    hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), str(valores[i])])
                    hola.setFlags(hola.flags() | Qt.ItemIsEditable)
                    self.listaHijos.append(hola)
                    self.listaHijos.append(elem)
                else:
                    hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), ""])
                    self.listaHijos.append(hola)
                    self.listaHijosStr.append(elem)
            j += 1
```

Figura 5.35: **Desarrollo del caso de uso GenerarInforme. Algoritmo para representar el árbol. Primera fila.**

```
# Filas distintas a la primera
else:
    j = 0
    for elem in fila:
        if j == (aux - 1):
            hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), str(valores[i])])
            hola.setFlags(hola.flags() | Qt.ItemIsEditable)
            self.listaHijos.append(hola)
            self.listaHijos.append(elem)

        else:
            if elem not in self.listaHijosStr:
                hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), ""])
                self.listaHijos.insert(j, hola)
                self.listaHijosStr.insert(j, elem)
        j += 1
i += 1
```

Figura 5.36: **Desarrollo del caso de uso GenerarInforme. Algoritmo para representar el árbol. Filas distintas a la primera.**

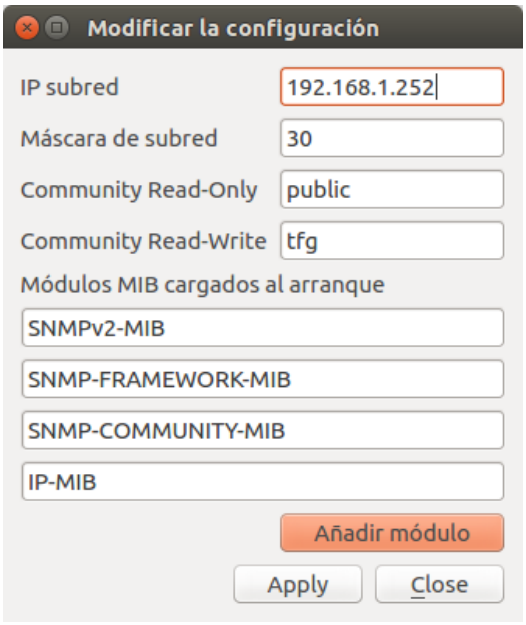

Figura 5.37: **Desarrollo del caso de uso ModificarConfiguración. Apariencia del diálogo**.

```
i = 0
self.listaCeldasMib = []
for item in self.mibUpload:
    self.celdaTemporal = QLineEdit(self.mibUpload[i])
    self.listaCeldasMib.insert(i, self.celdaTemporal)
    grid.addWidget(self.celdaTemporal, i+5, 0, 1, 2)
    i += 1
```

Figura 5.38: **Desarrollo del caso de uso ModificarConfiguración. Módulos MIB**.

igual al existente en el fichero de configuración en el momento de la lectura de `config.xml`. Existirá una quinta etiqueta, que ocupará todo el espacio horizontal equivalente a las duplas anteriores, con texto «Módulos MIB cargados al arranque». Tras esta etiqueta, se desplegarán tantas cajas de inserción de texto como módulos MIB existentes en el fichero de configuración. Un ejemplo de este diálogo puede verse en la figura 5.37.

En la figura 5.38 se muestra la parte del código relativa a representar un número inicialmente desconocido de campos destinados a los módulos MIB cargados al arranque, aunque el resto del código de la presente clase puede consultarse en el apéndice E (`editarConf.py`).

```
# Preparamos la escritura en nuestro fichero de configuración. Si alguno de los campos
# destinados a los módulos MIB se encuentra vacío, se eliminará del fichero de configuración.
for elem in self.tree.iter():
    if elem.tag == 'Subnet':
        elem.text = self.subnetIp
    elif elem.tag == 'Prefix':
        elem.text = str(self.subnetPrefix)
    elif elem.tag == 'CommunityRO':
        elem.text = self.communityRO
    elif elem.tag == 'CommunityRW':
        elem.text = self.communityRW
    elif elem.tag == 'MibUpload':
        i = 0
        for child in elem:
            if self.listaCeldasMib[i].text() is '':
                elem.remove(child)
            else:
                child.text = self.listaCeldasMib[i].text()
            i += 1

# Escribimos los cambios apropiados en el fichero de configuración y cerramos la ventana.
try:
    self.tree.write('config.xml')
    QMessageBox.information(self, "Configuración modificada", "Sus cambios han sido correctamente "
                                                              "escritos en el fichero de configuración.")
    self.close()
except Exception:
    QMessageBox.warning(self, "Error", "Su configuración no ha sido modificada debido a algún error.")
    self.close()
```

Figura 5.39: **Desarrollo del caso de uso ModificarConfiguración. Actualización del fichero**.

El usuario podrá ahora realizar los cambios que considere oportunos, pulsando «Àpply» cuando quiera que éstos tomen forma. Para ello, se llamará al método `apply`.

Esta método comprueba inicialmente que la dirección IP y el prefijo introducidos son correctos. Para ello, se sirve de dos métodos, `comprobarIp` y `comprobarPrefix` respectivamente, cuyo código puede verse en el apéndice E.

Una vez comprobado que el formato es el correcto, se procede a su actualización en `config.xml`, la cual podemos ver en la figura 5.39, finalizando por tanto el caso de uso que nos atañe.

5.2.8. Desarrollo del caso de uso DecodificarOID

Gracias a las facilidades ofrecidas por *PySNMP* podemos desarrollar este caso de uso sin que nos suponga demasiados quebraderos de cabeza. Tanto este caso de uso como el siguiente, han sido desarrollados completamente como parte de la clase `SnmpServer`. El fichero que la contiene, `snmpserver.py`, puede consultarse en el apéndice F.

```
for name, val in varBinds:
    name.resolveWithMib(self.mibViewController)
    labels.append(name.getLabel())
```

Figura 5.40: **Desarrollo del caso de uso DecodificarOID.** ***getLabel***.

```
########################################################################
#                  Método: decodificarOID                              #
#                                                                      #
# Este método se encarga de proporcionar acceso al método              #
# resolveWithMib del objeto mibViewController, instanciado en la       #
# inicialización de SnmpServer.                                        #
########################################################################
def decodificarOID(self, OID):
    return OID.resolveWithMib(self.mibViewController)
```

Figura 5.41: **Desarrollo del caso de uso DecodificarOID. Método** ***decodificarOID***.

Un requisito indispensable para la correcta satisfacción tanto del presente caso de uso como del siguiente, reside en la existencia de un compilador. Obtenemos este como instancia de la clase `MibCompiler`, dentro de la biblioteca `pysmi.compiler`. Además, será necesaria la instanciación de un `MibBuilder` y de un `MibViewController`, de las bibliotecas de *PySNMP* `pysnmp.smi.builder` y `pysnmp.smi.view` respectivamente.

Una vez realizadas estas instanciaciones, se decodifican los OIDs de dos maneras en el programa. La primera, como hemos visto para el caso de uso Ge, consiste en el método `getLabel` del `MibViewController`. Esto nos proporciona cada etiqueta por separado (es dnerarInformeecir, 'iso', 'org', etcétera) y lo empleamos para la representación en forma de árbol. El ejemplo del código podemos verlo en la figura 5.40.

La otra vía por la que resolvemos el OID, y que nos devuelve éste en formato módulo::etiqueta (por ejemplo, `SNMPv2-MIB::sysDescr.0`) es el método `decodificarOID` de la presente clase, que únicamente se encargará de, dado un OID, realizar la operación `resolveWithMib` del `MibViewController`, devolviendo el resultado de la operación y finalizando con ello el desarrollo del presente caso de uso, como podemos ver en la figura 5.41.

```
#####################################################################
#                      Método: loadMib                              #
#                                                                   #
# Este método se encarga de cargar al objeto mibBuilder los módulos #
# MIB pasados como entrada.                                         #
#####################################################################
def loadMib(self, *args):
    try:
        for arg in args:
            self.mibCompiler.compile(str(arg))
        return True
    except smi.error.MibNotFoundError:
        QMessageBox.warning(None, "Error SMI", "Alguna de las MIB pasadas para cargar no ha podido ser cargada.")
        return False
```

Figura 5.42: **Desarrollo del caso de uso CargarMIB**.

5.2.9. Desarrollo del caso de uso CargarMIB

A la hora del desarrollo de nuestro último caso de uso, y al haber cubierto buena parte del código del proyecto, no nos extenderemos demasiado.

Este caso de uso cumple las demandas de la secuencia de arranque para el cargado de módulos MIB y de la carga por parte del usuario en el momento de ejecución del programa. En la secuencia de arranque del programa (definida en el capítulo «Manual de uso»), todas las MIBs existentes en el fichero de configuración son cargadas haciendo uso del compilador descrito para el caso de uso previo. Una vez leído el fichero de configuración de la manera mostrada para otros casos de uso, como por ejemplo, AplicarConfiguración, se llama al método `loadMib` de la clase `SnmpServer`. Este método toma como entradas 0 o más módulos MIB a cargar y aprovecha el compilador mencionado para añadir estos a los módulos con los que el programa va a trabajar. Esta compilación es necesaria, aún teniendo instaladas las MIB en cuestión, para que los ficheros con los que trabajemos se encuentren en formato *Python*.

Este método también será empleado para las MIB solicitadas por el usuario en tiempo de ejecución. En este caso, habrán de ser introducidas una a una.

El método, que puede observarse en la figura 5.42, dispone de un control que determina si puede o no realizar la conversión. Nombres de módulos que no sean correctos o no puedan ser encontrados no serán compilados. Con el desarrollo de este método finalizamos el del caso de uso y, con él, cerramos el capítulo 5.

Capítulo 6

Pruebas

6.1. Introducción

Durante la fase de análisis recopilamos una serie de requisitos, funcionales y no funcionales, que debían ser satisfechos por nuestra aplicación, ya que respondían a la motivación y objetivos iniciales mostrados en el primer capítulo. Para ello, se desprendieron un conjunto de casos de uso, hasta un total de nueve, que pretendían abordar las distintas situaciones en que la aplicación debería desempeñarse a fin de dar cumplimiento a los requisitos recopilados. Estos casos de uso motivaron nuestro desarrollo, que fue mostrado en el capítulo precedente. Se pretende ahora demostrar que, tras la programación relativa a dichos casos de uso, se consigue dar satisfacción a todos los requisitos enunciados previamente.

Para ello, dedicaremos el presente capítulo al estudio y la demostración de los resultados obtenidos debido a la fase de pruebas, adecuando estos a los requisitos iniciales, divididos entre funcionales y no funcionales.

Como descripción de nuestro entorno de pruebas, en la red sobre la que nos aventuraremos a realizarlas existen dos elementos gestionados. En la dirección IPv4 192.168.1.13 disponemos de una impresora gestionable por SNMP, mientras que en la dirección 192.168.1.253 tenemos un demonio SNMP correctamente instalado y configurado en un *CentOS 7*.

Debemos añadir que, derivado del hecho de que determinados casos de uso satisfagan más de un requisito, habrá escenarios que impliquen el cum-

plimiento de más de un requisito. En el caso de que esto ocurra, mostraremos su resolución en el primero de los requisitos que analicemos que sea cumplido y haremos referencia a ello al llegar a otro requisito que haya sido cumplido con anterioridad.

6.2. Cumplimiento de requisitos

En la presente sección se pretende demostrar el cumplimiento de requisitos, dividiéndolos entre funcionales y no funcionales, como se ha explicado en la introducción del capítulo, y comenzando por los funcionales.

6.2.1. Requisitos funcionales

FRQ-001

El requisito FRQ-001 tenía la siguiente descripción: «El sistema deberá permitir al usuario establecer los valores que considere oportunos para aquellos OIDs de lectura/escritura» .

La operación a realizar en este caso, está claro, es un *set*. Para ello, debemos disponer de la *community* de lectura/escritura adecuada. Podemos especificarla en la base de datos para el elemento en concreto o bien emplear la *community* por defecto, existente en el fichero de configuración. El procedimiento que debemos seguir para poder cambiar el valor de un OID en un elemento gestionado es el siguiente:

1. Se debe seleccionar el elemento sobre el que queremos actuar de la ventana principal. En este caso, vamos a modificar un valor de nuestro *CentOS*.

2. Una vez realizado el doble *click* sobre el elemento en cuestión, se nos mostrará el diálogo que podemos ver en la figura 6.2. Para nuestro caso, hay tres opciones que nos acabarían sirviendo a nuestro fin. La más evidente es pulsar sobre el botón que nos indica «Cambiar parámetros», y será la primera que abordaremos en este caso.

3. Al seleccionar la opción mencionada, podemos ver que el diálogo se expande, mostrando dos campos destinados a la inserción de texto, inicialmente

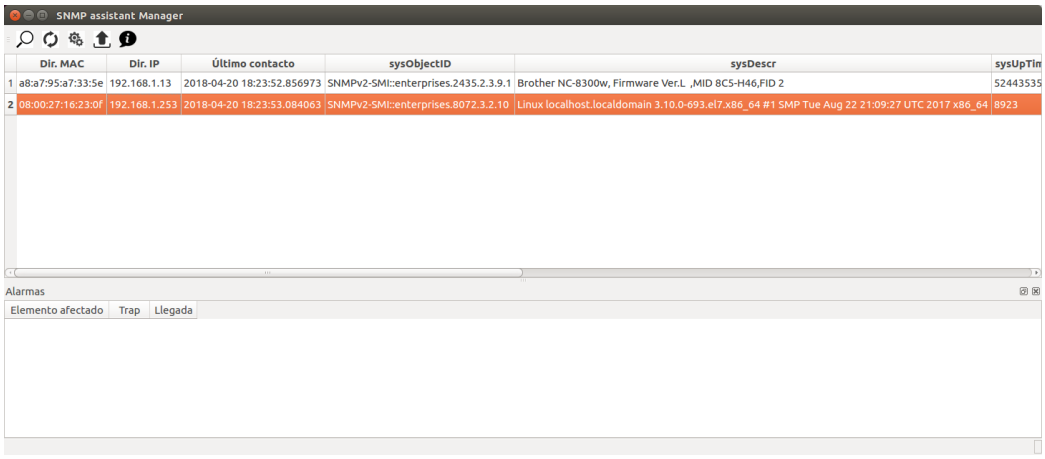

Figura 6.1: **Cumplimiento de FRQ-001. Selección de elemento**.

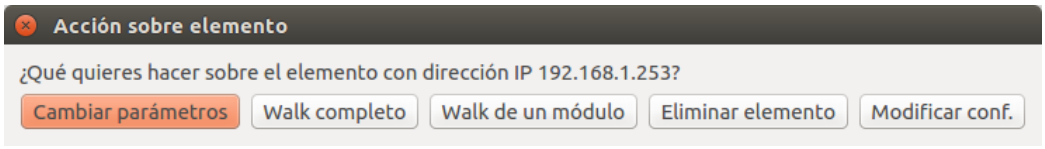

Figura 6.2: **Cumplimiento de FRQ-001. Cambiar parámetros**.

ocultos, como son el relativo al OID a modificar y el concerniente al nuevo valor. Vamos a alterar el parámetro *sysName* (1.3.6.1.2.1.1.5.0), inicialmente con valor «CentOS.valladoMIB», como podemos ver en la figura 6.3, para establecer el nombre «CentOS.Demostrado».

4. Una vez aceptado el cambio, el programa nos informa del resultado de la operación, que ha transcurrido con éxito, como también podemos ver en la ventana principal, con el cambio del campo *sysName* al que le habíamos indicado.

Como mencionamos previamente, existe otra alternativa para realizar la modificación de valor en un OID. Esta consistiría en alterar directamente el valor en la representación en forma de árbol mostrada para los requisitos FRQ-004 y NFRQ-002. De esta forma, el sistema trataría de realizar el *set* y el resultado sería el mismo que el obtenido por la vía que acabamos de describir.

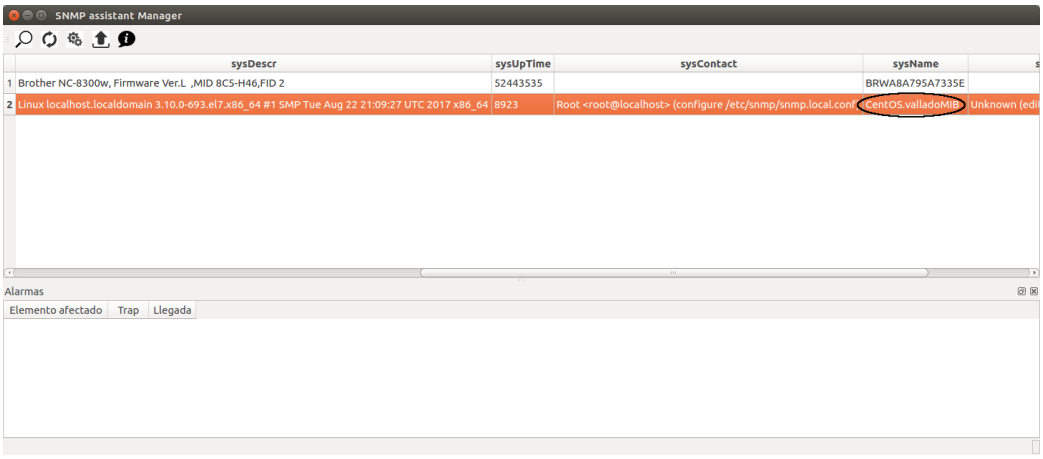

Figura 6.3: **Cumplimiento de FRQ-001. *sysName* original.**

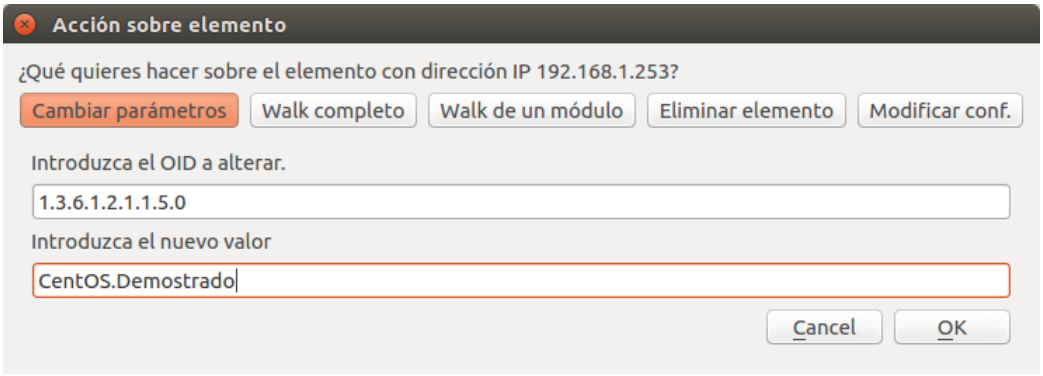

Figura 6.4: **Cumplimiento de FRQ-001. Introduciendo el cambio.**

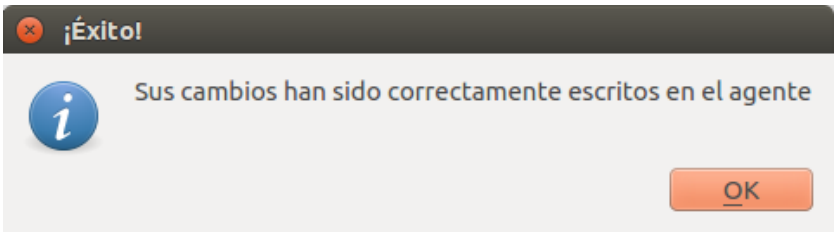

Figura 6.5: **Cumplimiento de FRQ-001. Éxito en el cambio.**

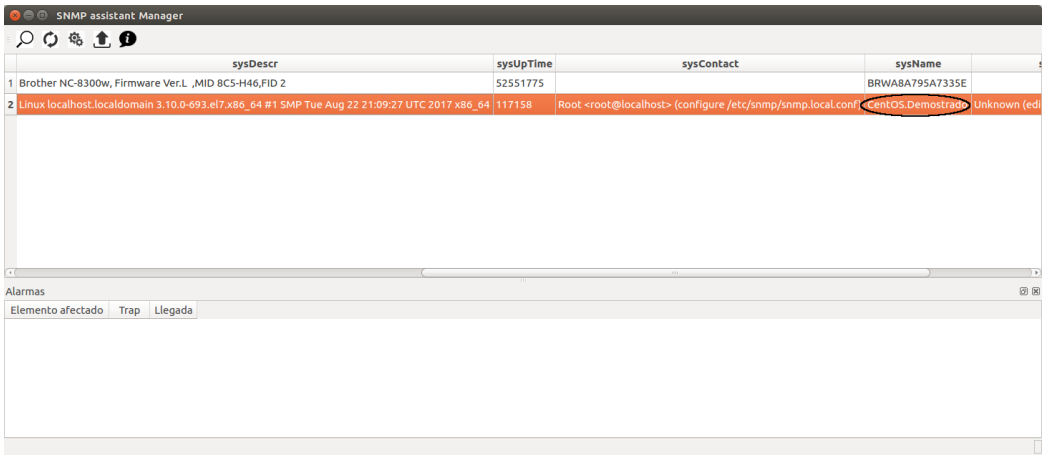

Figura 6.6: **Cumplimiento de FRQ-001. Valor cambiado**.

FRQ-002

La descripción del requisito FRQ-002 nos exige que «el sistema deberá permitir la notificación de eventos por parte de los elementos gestionados».

Para ello, en el capítulo previo, abordamos el desarrollo del caso de uso EnviarAlarma, que cubriría este requisito con su representación en pantalla.

Un caso típico en el que los agentes envían una notificación es, como vimos en el capítulo 2, un reinicio, tratado como *trap* del tipo *coldStart*. Vamos a forzar dicha «trampa». En la figura 6.7 podemos ver la interfaz gráfica inicial, previa a la recepción de la notificación, mientras que en la figura 6.8 mostramos cómo reiniciamos el demonio `snmpd` en la máquina que alberga el agente que enviará la «trampa».

El agente de nuestra máquina virtual *CentOS* está configurado para que, ante este hecho, envíe *traps* en versiones 1 y 2. Éstas podemos encontrarlas ahora en la tabla de alarmas de la parte inferior de la pantalla principal, coincidiendo la marca horaria y la dirección IP del elemento emisor de la notificación con las salidas obtenidas tras los comandos `ip addr show enp0s3` y `systemctl status snmpd`, respectivamente, en la máquina virtual.

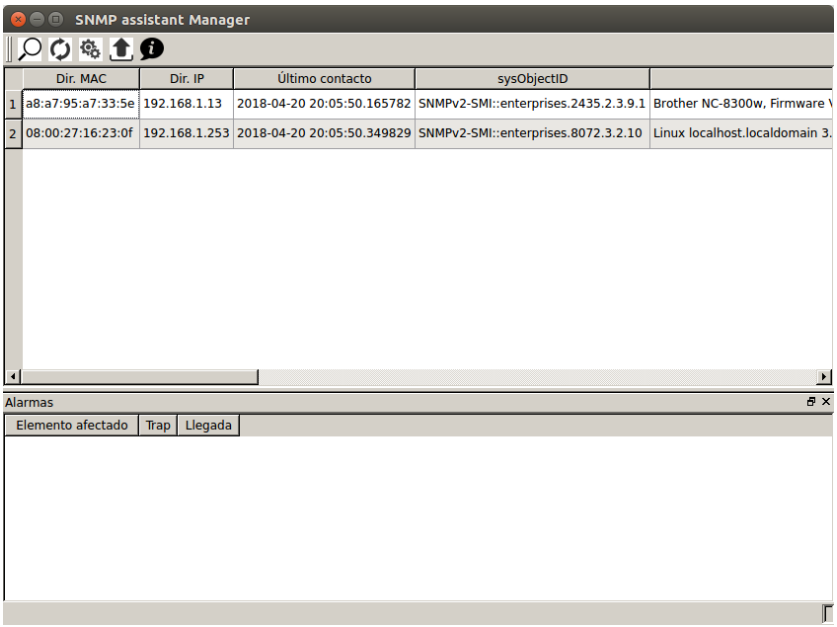

Figura 6.7: **Cumplimiento de FRQ-002. Ninguna notificación.**

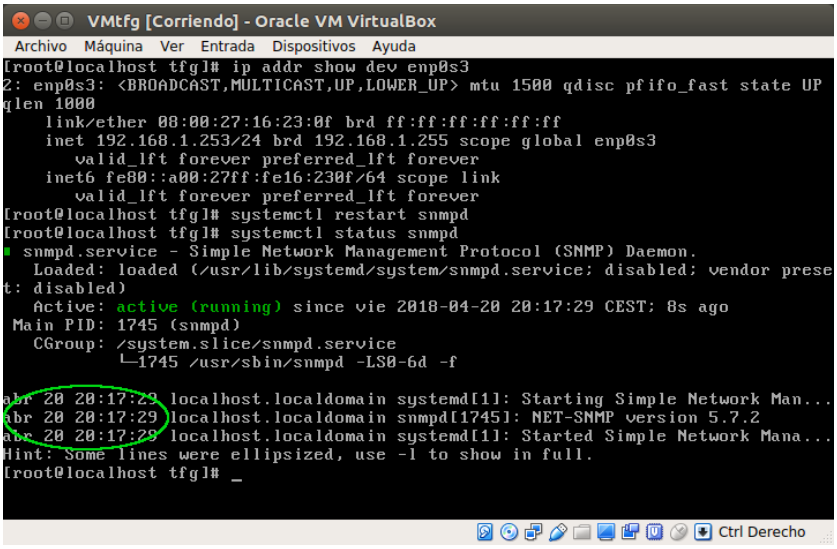

Figura 6.8: **Cumplimiento de FRQ-002. Reinicio del agente.**

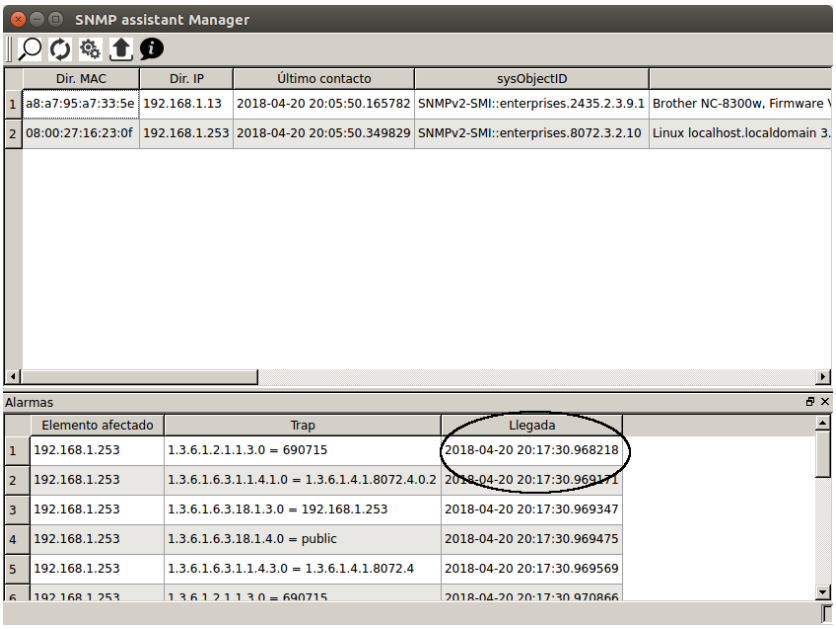

Figura 6.9: **Cumplimiento de FRQ-002. Alarmas representadas**.

FRQ-003

El requisito FRQ-003 afirma que «el sistema deberá permitir realizar búsquedas entre la base de datos de elementos gestionados».

La base de datos de elementos gestionados presenta una única tabla (`elementos`) que cuenta con cinco columnas, como puede verse en el capítulo 4, figura 4.4. Estas son, `mac_address`, `ip_address`, `lastTimeDetected`, `communityRO` y `communityRW`. Es por ello que el programa debería ofrecer una funcionalidad para satisfacer esta necesidad, con estos cinco parámetros como opciones para la búsqueda.

La solución la encontramos en la barra de herramientas, donde podemos encontrar el icono de una lupa, que nos llevará a tal funcionalidad.

Una vez seleccionada la opción de buscar, se nos mostrará un desplegable, expandible hasta un máximo de 8 campos, que nos permitirá realizar una búsqueda en la base de datos en función de distintas variables. Para nuestro ejemplo, vamos a seleccionar una búsqueda por 3 parámetros como **OR lógica** en la que el elemento buscado **contenga** 23:0f en su dirección MAC, 192.168 en su dirección IP y 2018 en último contacto. Así, nos debería

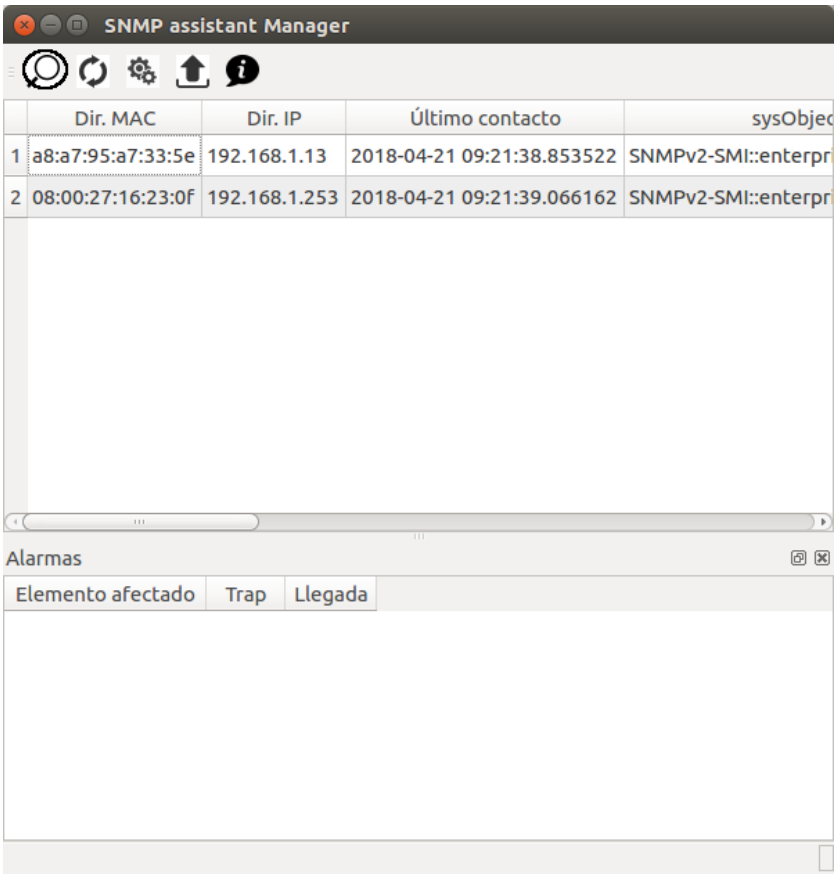

Figura 6.10: **Cumplimiento de FRQ-003. Icono de búsqueda**.

ser devuelta nuestra máquina virtual (por cumplir todas las condiciones) y nuestra impresora (por cumplir con la dirección IP y con el último descubrimiento). El resultado nos deberá mostrar también si el elemento tiene o no *communities* específicas, cosa que en nuestro caso no ocurrirá.

FRQ-004

En la descripción del requisito FRQ-004 nos encontramos con que «el sistema deberá permitir obtener datos de los clientes a petición del usuario». Vamos a demostrar que cumplimos con ello, solicitando al programa todos los datos relativos al módulo *SNMPv2-MIB::system* para la impresora gestionada.

Para ello, hacemos doble *click* sobre ella y, del diálogo que nos aparece, ya

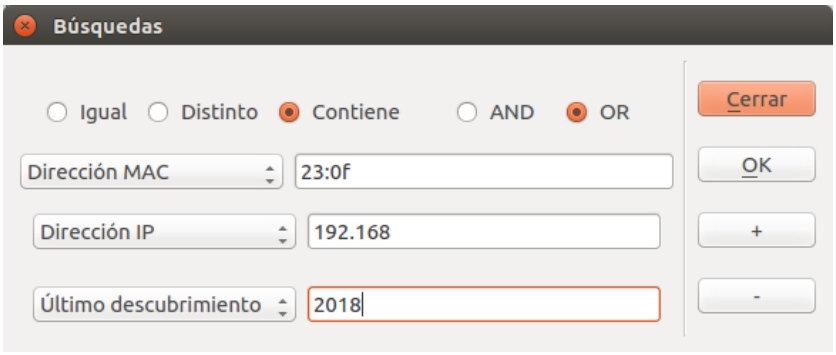

Figura 6.11: **Cumplimiento de FRQ-003. Parámetros de búsqueda.**

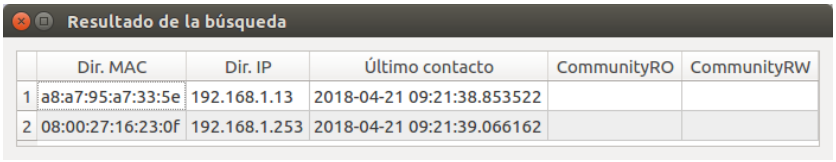

Figura 6.12: **Cumplimiento de FRQ-003. Resultado de la búsqueda.**

visto para el requisito FRQ-001, seleccionamos la opción *walk* de un módulo, insertando como módulo *SNMPv2-MIB*, como podemos observar en la figura 6.13.

El resultado de esta petición, representado en forma de árbol, se puede comprobar en la figura 6.14, cumpliendo con el presente requisito.

FRQ-005

Nuestro quinto requisito funcional establece que «el sistema deberá guardar en un fichero su configuración. Este fichero será leído cada vez que el

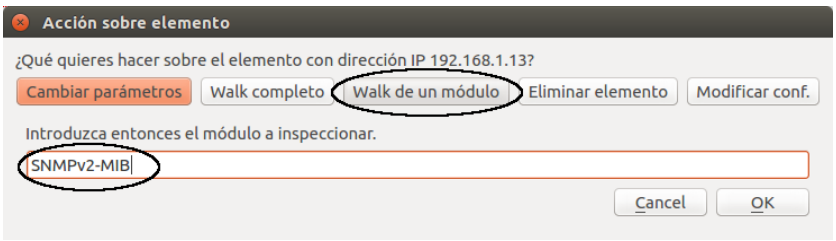

Figura 6.13: **Cumplimiento de FRQ-004. Petición.**

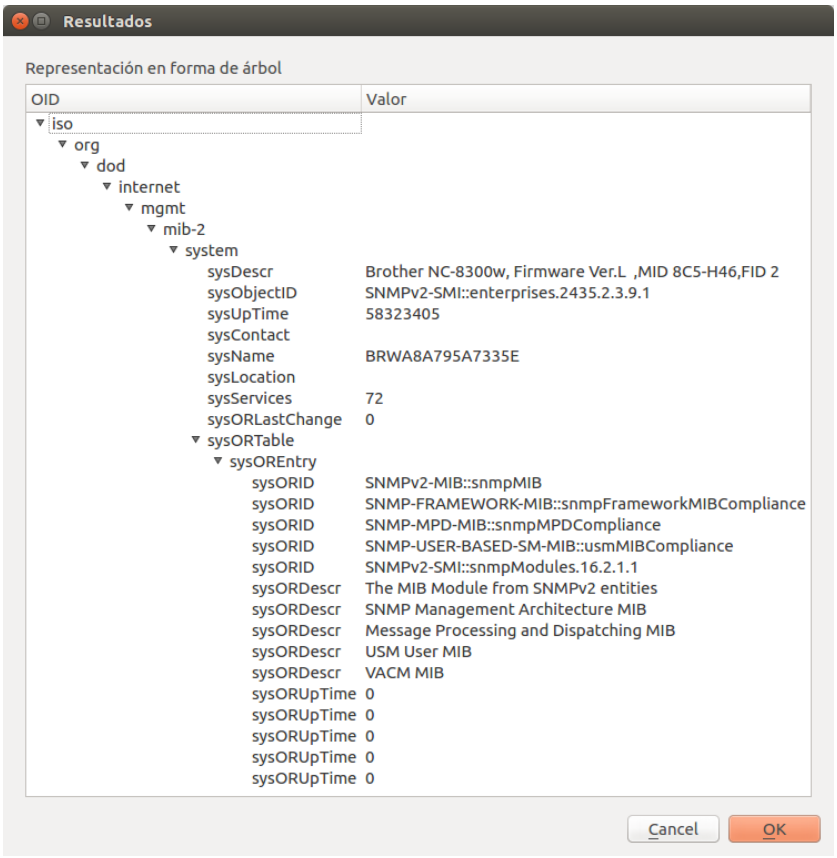

Figura 6.14: **Cumplimiento de FRQ-004. Resultados**.

sistema sea arrancado».

Para demostrar que, tras el desarrollo, se cumple con este requisito, vamos a mostrar dicho fichero de configuración (`config.xml`) (figura 6.14). Posteriormente, realizamos la operación que hemos mostrado para el FRQ-004 (*walk* de un módulo), pero sobre otro módulo. Concretamente, sobre `IP-MIB`, cargado al arranque gracias a la lectura del fichero de configuración (figura 6.15). El equipo sobre el que vamos a realizar la petición en esta ocasión es, nuevamente, la impresora (con dirección 192.168.1.13).

```
−<Config>
   <Subnet>192.168.1.252</Subnet>
   <Prefix>30</Prefix>
   <CommunityRO>public</CommunityRO>
   <CommunityRW>tfg</CommunityRW>
  −<MibUpload>
     <Mib1>SNMPv2-MIB</Mib1>
     <Mib2>SNMP-FRAMEWORK-MIB</Mib2>
     <Mib3>SNMP-COMMUNITY-MIB</Mib3>
     <Mib4>IP-MIB</Mib4>
   </MibUpload>
 </Config>
```

Figura 6.15: **Cumplimiento de FRQ-005.** ***config.xml***.

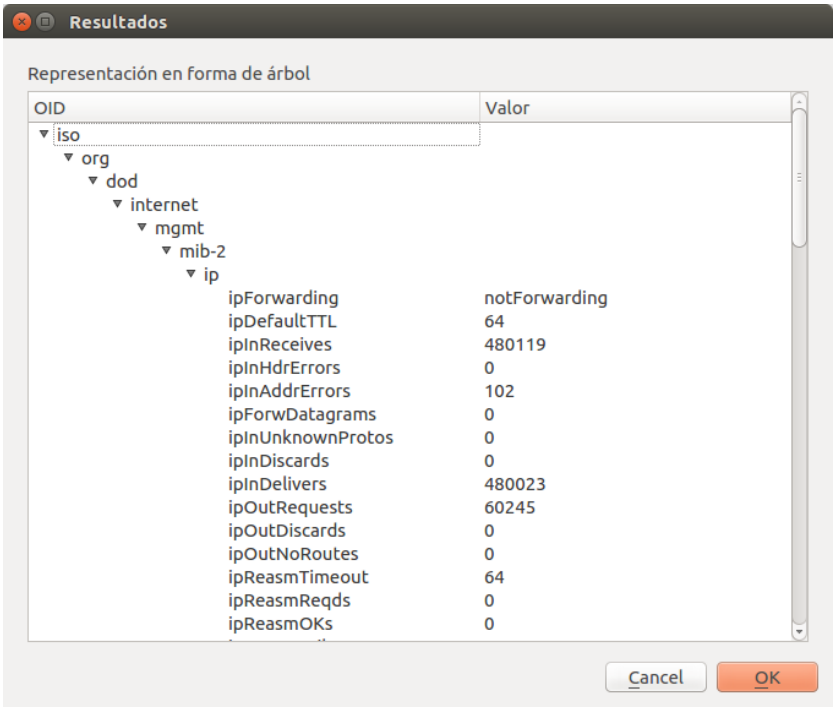

Figura 6.16: **Cumplimiento de FRQ-005. Resultado tras carga inicial**.

6.2.2. Requisitos no funcionales

NFRQ-001

El primero de nuestros requisitos no funcionales es el NFRQ-001, cuya descripción nos informa de que «el sistema deberá tratar las notificaciones de los equipos gestionados y mostrar al usuario alarmas en función de estos eventos».

A lo que nos referíamos con este requisito es que, la notificación o «trampa» enviada por el agente y recibida como parte del requisito FRQ-002, debía ser mostrada por pantalla. Es por ello que se desarrolló una sección de alarmas en la ventana principal, y su funcionamiento ya ha sido mostrado para justificar el cumplimiento del requisito FRQ-002, por lo que podríamos acudir a la figura 6.9 para contemplar la representación de las alarmas de que hablamos.

NFRQ-002

Al igual que en el requisito previo, en este caso también hemos demostrado indirectamente el cumplimiento de las exigencias, que afirman que «el sistema deberá poder listar la información correspondiente a los equipos gestionados en forma de árbol».

En este caso nos referimos al requisito FRQ-004. La estructura SMI, como pudimos ver en el capítulo 2, provoca una clara motivación a que la información de un elemento sea mostrada en forma de árbol. Ejemplos de esta forma de representación los encontramos en las figuras 6.14 y 6.16.

NFRQ-003

El requisito NFRQ-003 solicita que «el sistema deberá decodificar los OIDs a fin de presentar al usuario la información en un formato amigable».

En este caso, de nuevo, hemos cumplido el requisito indirectamente gracias a la resolución de FRQ-004. La decodificación de OID es llevada a cabo en la representación en forma de árbol, de manera que, en lugar de mostrar *sysObjectID* como 1.3.6.1.2.1.1.2.0, se muestra, precisamente, como *sysOb-*

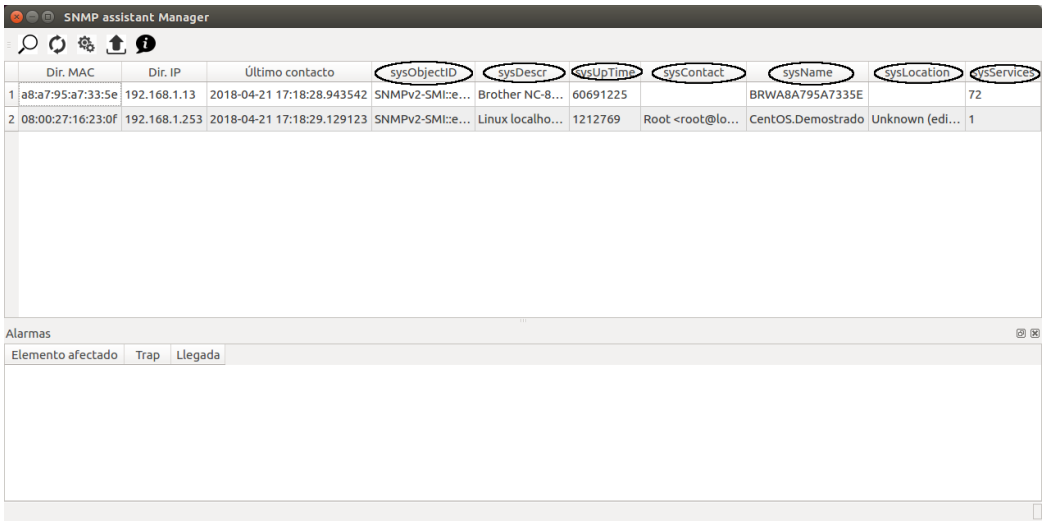

Figura 6.17: **Cumplimiento de NFRQ-003. Decodificación en la ventana principal**.

jectID con sus nodos precedentes también decodificados (como *iso*, *org*, etcétera). Esto podemos verlo en las figuras 6.14 y 6.16, como en el caso anterior, pero también en la representación de elementos gestionados en la ventana principal, en los que mostramos una serie de parámetros SNMP, entre los que se encuentran *sysObjectID*, *sysDescr*, *sysUpTime*, *sysContact*, *sysName*, *sysLocation* y *sysServices*, como podemos observar en la figura 6.17.

NFRQ-004

El requisito NFRQ-004 indica que «el sistema deberá permitir al usuario alterar el fichero de configuración de la aplicación. El acceso a dicho fichero será proporcionado a través de la interfaz gráfica».

Procedemos a mostrar un cambio en dicho fichero. Para ello, el usuario deberá seleccionar la opción «Editar configuración», identificada mediante un engranaje en la barra de herramientas.

La interfaz existente para alterar el citado fichero puede verse en la figura 6.19. Al inicio, es mostrada con los valores presentes en ese momento en el fichero de configuración. Es por ello que, para demostrar que podemos cambiarla, vamos a modificar la IP de subred, inicialmente con valor 192.168.1.252, a 192.168.1.248; y la máscara de subred, que se encuentra

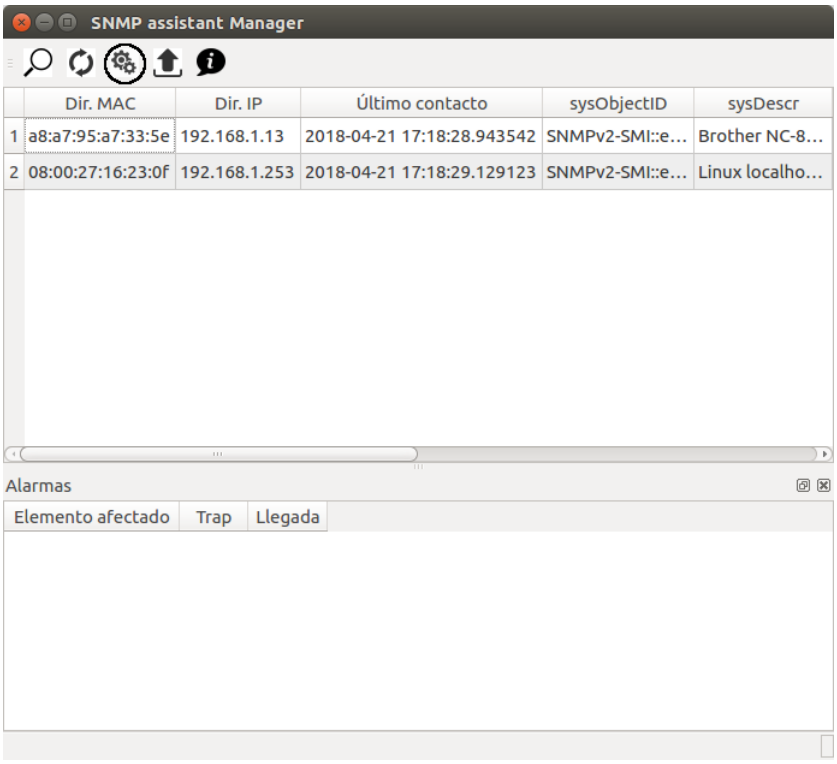

Figura 6.18: **Cumplimiento de NFRQ-004. Opción «Editar configuración».**

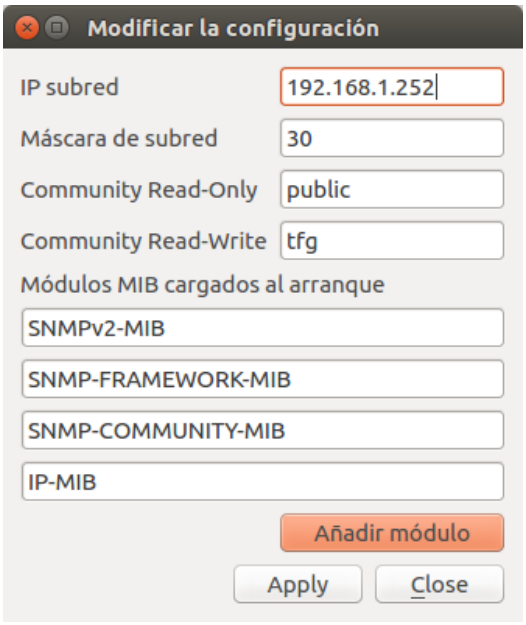

Figura 6.19: **Cumplimiento de NFRQ-004. Interfaz de edición**.

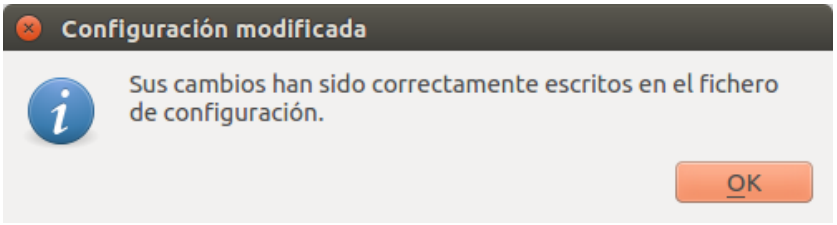

Figura 6.20: **Cumplimiento de NFRQ-004. Configuración modificada**.

inicialmente con valor 30 bits y la vamos a dejar con 29.

Tras escribir los cambios y pulsar en «Apply», se nos mostrará el resultado de la operación, que podemos comprobar en la figura 6.20.

Los resultados podemos comprobarlos, bien abriendo directamente el fichero `config.xml`, bien volviendo a *clickar* sobre el icono indicado. En ambos casos veremos que los cambios han tenido efecto.

```
−<Config>
   <Subnet>192.168.1.248</Subnet>
   <Prefix>29</Prefix>
   <CommunityRO>public</CommunityRO>
   <CommunityRW>tfg</CommunityRW>
  −<MibUpload>
     <Mib1>SNMPv2-MIB</Mib1>
     <Mib2>SNMP-FRAMEWORK-MIB</Mib2>
     <Mib3>SNMP-COMMUNITY-MIB</Mib3>
     <Mib4>IP-MIB</Mib4>
   </MibUpload>
 </Config>
```

Figura 6.21: **Cumplimiento de NFRQ-004. Cambios mostrados en *config.xml*.**

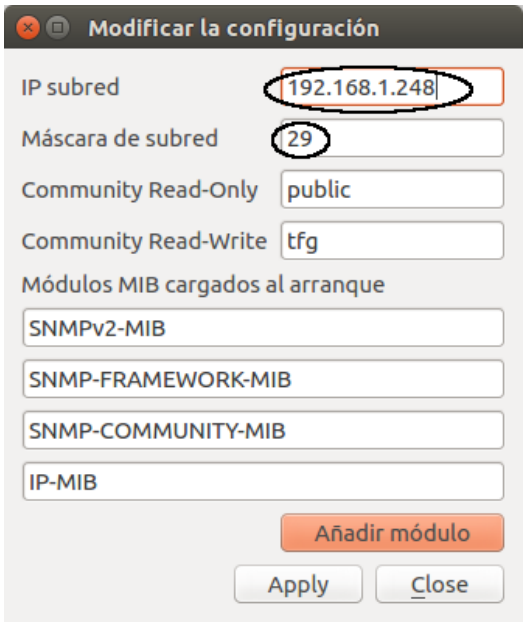

Figura 6.22: **Cumplimiento de NFRQ-004. Cambios mostrados en la interfaz gráfica.**

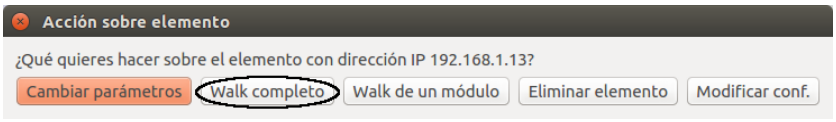

Figura 6.23: **Cumplimiento de NFRQ-005. *Walk* completo.**

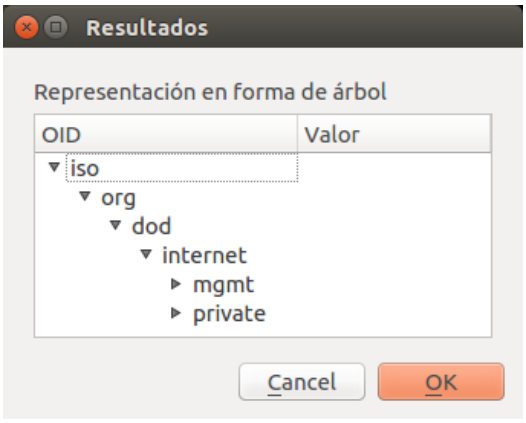

Figura 6.24: **Cumplimiento de NFRQ-005. Resultados compactos.**

NFRQ-005

El requisito NFRQ-005 enuncia que «el sistema deberá permitir mostrar información detallada de un cliente concreto a petición del usuario».

Hemos podido ver un acercamiento en varios requisitos hacia este punto, aunque el propósito real comprende el hecho de tener un informe completo de todos los objetos accesibles de un agente. A fin de lograrlo, se han desarrollado los mecanismos necesarios para obtener un *walk* completo del agente en cuestión. Para ello, se debe realizar doble *click* sobre el dispositivo que aparece en la pantalla principal y, en el diálogo posterior, se habrá de elegir la opción «Walk completo».

Una vez hecho esto, sobre la impresora con dirección IP 192.168.1.13, obtenemos el resultado que mostramos en la figura 6.24. Al existir miles de OIDs gestionables para este elemento, el resultado, en forma de árbol, se muestra compactado.

NFRQ-006

El requisito NFRQ-006 indica que «la versión a utilizar será v2c».

Este hecho lo hemos visto de manera transversal durante todo el capítulo, pues el uso de *communities* como método de autenticación es empleado para satisfacer prácticamente todos los requisitos.

Capítulo 7

Conclusiones y líneas futuras

El capítulo que comienza finaliza el documento. Dividido en dos secciones, pretende dar un colofón para todo el contenido previo. Comenzaremos recopilando una serie de conclusiones, generales y personales, que quien tiene estas hojas entre manos podrá haber ido entreviendo a lo largo del contenido precedente, terminando la memoria con un conjunto de líneas futuras sobre las que trabajar más allá de este trabajo.

7.1. Conclusiones

Nos acercamos al cierre del presente trabajo fin de grado. Una vez finalizada la lectura del documento el lector habrá podido obtener una serie de valiosas conclusiones acerca, primero, del estado del arte, ya no sólo del protocolo SNMP en particular, sino del mundo de gestión de redes en general, y, segundo, de las funcionalidades y facilidades que una aplicación de estas características ofrece al sector.

Estas conclusiones son, por tanto, las que se han intentado plasmar a lo largo de todo el documento, aglutinadas como parte del capítulo final.

Pese al ánimo general de innovación en el mundo de Internet, la demora en el desarrollo del modelo OSI lo condenó debido al alto grado de penetración de TCP/IP en el momento en que se pretendió dar la transición esperada. De esta situación se benefició SNMP, convirtiéndose en el protocolo más empleado para la gestión de redes durante, por el momento, una treintena

de años.

Una larga retahíla de fabricantes se lanzó a vender sus propias soluciones de gestión de red basadas en SNMP, potentes, eficaces y con alto grado de penetración en el mercado a día de hoy. No obstante, no existe una aplicación de gestión SNMP y manejo de MIB libre y ambiciosa hoy día. Este trabajo fin de grado no llega a ofrecer una herramienta lo suficientemente potente para los deseos del alumno, pero sin duda supone una alternativa de calidad a las soluciones gratuitas existentes, abierta a futuras modificaciones y mejoras.

Además, el hecho de estar escrita en *Python*, un lenguaje de programación en claro auge, cuyo uso se estima ya a décimas porcentuales de *Java*, que continúa siendo el más popular en marzo de 2018, deja a la aplicación preparada para un mundo en que el lenguaje creado por Guido Van Rossum será, cuanto menos, una de las vanguardias.

Conclusiones personales

Para el estudiante, este Trabajo de Fin de Grado le ha permitido entrar en contacto con distintas competencias y metodologías adquiridas a lo largo del grado, por lo que se considera altamente satisfactorio en lo personal.

Más allá del mundo de la gestión de red más puro, con asignaturas del grado como Administración y Gestión de Redes de Comunicaciones o Laboratorio Avanzado de Redes y Servicios Telemáticos, se han aplicado conceptos de ingeniería de software (vistos en Ingeniería de Sistemas Software), gestión de bases de datos y tecnologías web como XML (estudiado en Tecnologías de Aplicaciones Web), conceptos básicos y avanzados de la programación orientada a objetos (complementando lo estudiado en Programación, Ingeniería de Sistemas Software, Arquitectura de Aplicaciones Distribuidas o Redes de Comunicación Avanzadas, entre otras) e, incluso, aspectos relativos a asignaturas desconocidas por el estudiante debido a su optatividad, como es el caso de la programación multiproceso que puede aprenderse en Sistemas de Tiempo Real. Todo esto ha sido posible gracias al aprendizaje de conceptos relacionados con asignaturas que fueron la base de todo el conocimiento personal sobre la telemática, como puede ser el caso de Arquitectura de Redes, Sistemas y Servicios, Ingeniería de Protocolos o Conmutación y Encaminamiento.

La experiencia laboral, aún corta aunque enriquecedora, en el mundo

de la gestión de redes en mi etapa de prácticas en Michelin y ahora como trabajador en Nokia, me ha ayudado a dar perspectiva a las necesidades reales existentes en la industria, no haciendo más que reafirmar la necesidad de soluciones como la aportada por este proyecto.

Por último, el aprendizaje de la programación en *Python*, lenguaje que no conocía hasta el momento de realización del TFG, me parece un valor clave y estratégico en el futuro desempeño profesional de un mundo que, como hemos afirmado en distintos párrafos, será dominado, casi con total seguridad, por la serpiente.

7.2. Líneas futuras

Como hemos adelantado en el apartado de conclusiones, la aplicación mira hacia el futuro, con múltiples posibles mejoras que, con seguridad, harán de ésta una aplicación potente y referente.

El hecho de estar escrita en *Python 3.5*, una versión actualizada de un lenguaje en proyección meteórica, hace que nuestra herramienta tenga todo a su favor para caminar hacia delante.

Entre las posibles mejoras y líneas futuras que admite nuestro programa, podemos destacar las siguientes:

- **Registro de la herramienta**. Tras ver la falta de alternativas en esta dirección y lo conseguido, parece que el siguiente paso a tomar no es otro que registrar la herramienta e intentar añadirla a los repositorios oficiales. Esto es algo que, seguro, intentaremos realizar en los meses siguientes a la defensa del presente Trabajo Fin de Grado.

- **Gestión de alarmas**. El prototipo inicial propone una gestión de alarmas en las que únicamente se muestra al usuario la *trap* recibida, sin extraer más información de ella ni realizar ninguna operación adicional. En este aspecto, resultaría interesante proporcionar persistencia en las alarmas y ofrecer al usuario un mecanismo de gestión de ellas más allá de, simplemente, informar de las notificaciones recibidas.

- **Manejo de nuevas opciones de autenticación y seguridad**. Nuestro programa trabaja con una autenticación basada en *communities*,

realizando las peticiones del servidor en versión 2c. Puede hacerse necesario trabajar con agentes que no soporten este tipo de operación por lo que sería bueno avanzar en el desarrollo del código necesario para soportar los envíos en versión 3, con las características que USM y VACM conllevan. Además, también podría ser interesante el añadido de seguridad criptográfica para, al menos, dar al usuario la opción de emplearla desde el programa.

- **El paso a IPv6**. Aunque *PySNMP* se encuentra preparado para el paso a IPv6, no se ha contemplado en nuestra aplicación. La demora con que éste se está dando no implica que su llegada no vaya a producirse. De hecho, la penetración del IoT (*Internet of Things*) y el inminente aterrizaje del 5G (que ya toma forma en Segovia y Talavera de la Reina de la mano de Telefónica, Nokia y Ericsson [60]) dan aún más motivos para la asimilación de esta versión por Internet en general, y, por extensión, por nuestra herramienta.

Por todo esto, podemos concluir afirmando que el desarrollo de esta aplicación cubre una necesidad previa pero, desde luego, no finaliza con la presente memoria, sino que puede y debe seguir un camino de ambición y adecuación a los nuevos tiempos que vengan.

Apéndice A

dialogosAdicionales.py

```
#!/usr/bin/env python3.5
# -*- coding: utf-8 -*-
# Fichero: dialogosAdicionales.py
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
###############################################################################
# Descripción: El objetivo del siguiente fichero es agrupar los               #
# distintos diálogos no invocados mediante alguna de las opciones de la       #
# barra de herramientas.                                                      #
###############################################################################

# Importaciones generales
import sys

# Importaciones de PyQT
from PyQt4.QtCore import *
from PyQt4.QtGui import *

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
###############################################################################
#                          Historial de versiones                             #
# ----------------------------------------------------------------------------#
# Versión 1.0.0: 22 de marzo de 2018                                          #
#                  Versión inicial                                            #
# Versión 1.0.1: 25 de marzo de 2018                                          #
#                  Manejo de cambios en el árbol debido al usuario.           #
# Versión 1.0.2: 1 de abril de 2018                                           #
#                  Opción de editar configuración de elemento.                #
#                  Opción de eliminar elemento.                               #
# Versión 1.1.0: 10 de abril de 2018                                          #
#                  Primera versión totalmente operativa.                      #
###############################################################################

###############################################################################
#                          Clase: SetWalkNextDlg                              #
#                                                                             #
# Clase encargada de mostrar un diálogo en que se pueda elegir entre          #
# realizar un walk de un elemento al completo, de uno de sus módulos,         #
# o bien, alterar alguno de sus parámetros. Es invocado tras clickar          #
# sobre un elemento de la tabla de elementos gestionados de la ventana        #
# principal. Presenta elementos ocultos para insertar el módulo sobre         #
# el que operar (en el caso de walk de un módulo) o el OID y el nuevo         #
# valor (en el caso de un set). Tras la interacción con el usuario,           #
# alterará sus parámetros adecuadamente, que serán accedidos desde la         #
# ventana principal.                                                          #
###############################################################################
class SetWalkNextDlg(QDialog):

    # Inicialización
    def __init__(self, mac, ip, parent=None):
        super(SetWalkNextDlg, self).__init__(parent)
```

```
        # Estos tres parámetros públicos serán alterados en virtud de las acciones
        # del usuario y serán accedidos desde MainWindow(QMainWindow) para invocar
        # las distintas acciones del servidor.
        self.getNextOrdenado = False
        self.walkOrdenado = False
        self.setOrdenado = False
        self.deleteOrdenado = False
        self.editConfOrdenada = False

        # Guardamos como atributos públicos de clase las direcciones MAC e IP pasadas en
     la inicialización
        self.mac = mac
        self.ip = ip

        # Etiqueta. Recibimos la dirección IP de MainWindow.
        texto = "¿Qué quieres hacer sobre el elemento con dirección IP " + self.ip + "?"
        etiqueta1 = QLabel(texto)

        # Botonera. El set y el getnext son "Checkable" para que su botón sirva para
    mostrar y ocultar
        self.snmpset = QPushButton("Cambiar parámetros")
        self.snmpset.setCheckable(True)
        self.snmpwalk = QPushButton("Walk completo")
        self.snmpgetnext = QPushButton("Walk de un módulo")
        self.snmpgetnext.setCheckable(True)
        self.borrarElemento = QPushButton("Eliminar elemento")
        self.modificarConfig = QPushButton("Modificar conf.")
        self.modificarConfig.setCheckable(True)
        botonera = QHBoxLayout()
        botonera.addWidget(self.snmpset)
        botonera.addWidget(self.snmpwalk)
        botonera.addWidget(self.snmpgetnext)
        botonera.addWidget(self.borrarElemento)
        botonera.addWidget(self.modificarConfig)

        # Sección inicialmente oculta para el getNext
        self.ocultoNext = QFrame()
        nextLayout = QVBoxLayout()
        etiquetaNext = QLabel("Introduzca entonces el módulo a inspeccionar.")
        self.insercionModulo = QLineEdit("Introduzca aquí el módulo deseado...")
        self.buttonGetNext = QDialogButtonBox(QDialogButtonBox.Cancel | QDialogButtonBox
    .Ok)
        nextLayout.addWidget(etiquetaNext)
        nextLayout.addWidget(self.insercionModulo)
        nextLayout.addWidget(self.buttonGetNext)
        self.ocultoNext.setLayout(nextLayout)

        # Sección inicialmente oculta para el set
        self.ocultoSet = QFrame()
        self.snmpsetLayout = QVBoxLayout()
        etiquetaSet1 = QLabel("Introduzca el OID a alterar.")
        self.recogerOid = QLineEdit("Introduzca aquí el OID...")
        etiquetaSet2 = QLabel("Introduzca el nuevo valor")
        self.recogerNuevoValor = QLineEdit("Introduzca aquí el valor deseado...")
        self.buttonSetBox = QDialogButtonBox(QDialogButtonBox.Cancel | QDialogButtonBox.
    Ok)
        self.snmpsetLayout.addWidget(etiquetaSet1)
        self.snmpsetLayout.addWidget(self.recogerOid)
        self.snmpsetLayout.addWidget(etiquetaSet2)
        self.snmpsetLayout.addWidget(self.recogerNuevoValor)
        self.snmpsetLayout.addWidget(self.buttonSetBox)
        self.ocultoSet.setLayout(self.snmpsetLayout)

        # Sección inicialmente oculta para modificar configuración
        self.ocultoConfig = QFrame()
        self.modificarConfigLayout = QVBoxLayout()
        etiquetaConfig1 = QLabel("Community de sólo lectura")
        self.recogerCommRO = QLineEdit("Introduzca la nueva communityRO del elemento")
        etiquetaConfig2 = QLabel("Community de lectura/escritura")
        self.recogerCommRW = QLineEdit("Introduzca la nueva communityRW del elemento")
        self.buttonConfigBox = QDialogButtonBox(QDialogButtonBox.Cancel |
    QDialogButtonBox.Ok)
        self.modificarConfigLayout.addWidget(etiquetaConfig1)
        self.modificarConfigLayout.addWidget(self.recogerCommRO)
        self.modificarConfigLayout.addWidget(etiquetaConfig2)
        self.modificarConfigLayout.addWidget(self.recogerCommRW)
        self.modificarConfigLayout.addWidget(self.buttonConfigBox)
        self.ocultoConfig.setLayout(self.modificarConfigLayout)

        # Construimos el layout principal...
        mainLayout = QVBoxLayout()
        mainLayout.addWidget(etiqueta1)
        mainLayout.addLayout(botonera)
```

```
        mainLayout.addWidget(self.ocultoNext)
        mainLayout.addWidget(self.ocultoSet)
        mainLayout.addWidget(self.ocultoConfig)
        self.setLayout(mainLayout)

        # Escondemos las secciones adicionales para set y getNext
        self.ocultoNext.hide()
        self.ocultoSet.hide()
        self.ocultoConfig.hide()

        # Hacemos fijo el tamaño para que no se deforme tras la interacción con los
    elementos ocultos
        mainLayout.setSizeConstraint(QLayout.SetFixedSize)

        # Título de la ventana
        self.setWindowTitle("Acción sobre elemento")

        # Conexiones de la botonera
        self.connect(self.snmpgetnext, SIGNAL("toggled(bool)"), self.ocultoNext, SLOT("
    setVisible(bool)"))
        self.connect(self.snmpset, SIGNAL("toggled(bool)"), self.ocultoSet, SLOT("
    setVisible(bool)"))
        self.connect(self.modificarConfig, SIGNAL("toggled(bool)"), self.ocultoConfig,
    SLOT("setVisible(bool)"))
        self.connect(self.buttonSetBox, SIGNAL("accepted()"), self.acceptSet)
        self.connect(self.buttonSetBox, SIGNAL("rejected()"), self.cancelSet)
        self.connect(self.snmpwalk, SIGNAL("clicked()"), self.orderWalk)
        self.connect(self.buttonGetNext, SIGNAL("accepted()"), self.acceptGetNext)
        self.connect(self.buttonGetNext, SIGNAL("rejected()"), self.cancelGetNext)
        self.connect(self.borrarElemento, SIGNAL("clicked()"), self.confirmarBorrar)
        self.connect(self.buttonConfigBox, SIGNAL("accepted()"), self.acceptEditConfig)
        self.connect(self.buttonConfigBox, SIGNAL("rejected()"), self.cancelEditConfig)


    ##########################################################################
    #                          Método: acceptSet                             #
    #                                                                        #
    # Recoge el click del usuario en el botón Ok del Set, altera el          #
    # parámetro público setOrdenado y cierra el diálogo.                     #
    ##########################################################################
    def acceptSet(self):
        self.setOrdenado = True
        self.close()


    ##########################################################################
    #                          Método: cancelSet                             #
    #                                                                        #
    # Recoge el click del usuario en el botón Cancel del Set y oculta        #
    # de nuevo esta parte del diálogo.                                       #
    ##########################################################################
    def cancelSet(self):
        self.ocultoSet.hide()


    ##########################################################################
    #                          Método: orderWalk                             #
    #                                                                        #
    # Recoge el click del usuario en el botón "Walk completo", altera        #
    # el parámetro público walkOrdenado y cierra el diálogo.                 #
    ##########################################################################
    def orderWalk(self):
        self.walkOrdenado = True
        self.close()


    ##########################################################################
    #                          Método: acceptGetNext                         #
    #                                                                        #
    # Recoge el click del usuario en el botón Ok del GetNext, altera         #
    # el parámetro público getNextOrdenado y cierra el diálogo.              #
    ##########################################################################
    def acceptGetNext(self):
        self.getNextOrdenado = True
        self.close()


    ##########################################################################
    #                          Método: cancelGetNext                         #
    #                                                                        #
    # Recoge el click del usuario en el botón Cancel del GetNext y           #
    # oculta de nuevo esta parte del diálogo.                                #
    ##########################################################################
```

```
    def cancelGetNext(self):
        self.ocultoNext.hide()


    ##########################################################################
    #                          Método: confirmarBorrar                       #
    #                                                                        #
    # Ante la selección del usuario de borrar el elemento, solicitamos       #
    # confirmación. En caso de ser proporcionada, establecemos la            #
    # variable de control a True y cerramos el diálogo.                      #
    ##########################################################################
    def confirmarBorrar(self):
        confir = QMessageBox.warning(self, "Borrado de elemento", "¿Está seguro de que
     quiere eliminar el elemento con"
                                                                   "dirección IP " + self.ip + " y
      dirección MAC " + self.mac +
                                                                    " de la base de datos?",
     QMessageBox.Ok, QMessageBox.Cancel)
        if confir == QMessageBox.Ok:
            self.deleteOrdenado = True
            self.close()


    ##########################################################################
    #                       Método: acceptEditConfig                         #
    #                                                                        #
    # Ante el 'click' del usuario sobre el botón 'Ok' para editar            #
    # configuración, ponemos a True la variable de control y cerramos        #
    # el diálogo.                                                            #
    ##########################################################################
    def acceptEditConfig(self):
        self.editConfOrdenada = True
        self.close()


    ##########################################################################
    #                       Método: cancelEditConfig                         #
    #                                                                        #
    # Ante el 'click' del usuario sobre el botón 'Cancelar', ocultamos       #
    # de nuevo esta parte del diálogo.                                       #
    ##########################################################################
    def cancelEditConfig(self):
        self.ocultoConfig.hide()


##############################################################################
#                              Clase: DlgArbol                               #
#                                                                            #
# Esta clase se encarga de mostrar una representación en forma de            #
# árbol. Será instanciada tras la realización de un Walk o similar y         #
# toma como entradas para su inicialización una lista de valores de los      #
# elementos y una lista de listas con las etiquetas (por ejemplo, iso,       #
# dod, org, system...) y se encargará de relacionar ambas en un árbol        #
# de dos columnas. También se encargará de manejar los posibles cambios      #
# que el usuario pueda introducir.                                           #
##############################################################################
class DlgArbol(QDialog):

    # Inicialización
    def __init__(self, valores, etiquetas, parent=None):
        super(DlgArbol, self).__init__(parent)

        # Parámetros públicos de la clase. Todos salvo listaHijosStr serán accedidos por
      MainWindow
        # para gestionar los cambios introducidos por el usuario.
        self.cambios = []
        self.cambiosRutas = []
        self.controlCambio = False
        self.listaHijosStr = []

        # Elementos del diálogo. El widget que nos servirá para representar el árbol es
     QTreeWidget
        treeLabel = QLabel("Representación en forma de árbol")
        self.treeWidget = QTreeWidget()
        treeLabel.setBuddy(self.treeWidget)
        splitter = QSplitter(Qt.Horizontal)

        buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        vbox = QVBoxLayout()
        vbox.addWidget(treeLabel)
        vbox.addWidget(self.treeWidget)
```

```
        widget = QWidget()
        widget.setLayout(vbox)
        splitter.addWidget(widget)

        layout = QVBoxLayout()
        layout.addWidget(splitter)
        layout.addWidget(buttonBox)
        self.setLayout(layout)

        # Llamamos al método populateTree con los valores pasados en la instanciación
        # para representar el árbol de manera adecuada.
        self.populateTree(valores, etiquetas)

        self.setWindowTitle("Resultados")

        # Creamos las conexiones para gestionar el cambio de valor en un elemento y los
        # botones Ok y Cancel.
        self.connect(self.treeWidget, SIGNAL("itemChanged(QTreeWidgetItem *, int)"),
    self.recogerCambios)
        self.connect(buttonBox.button(QDialogButtonBox.Ok), SIGNAL("clicked()"), self.
    pulsaronOk)
        self.connect(buttonBox.button(QDialogButtonBox.Cancel), SIGNAL("clicked()"),
    self.pulsaronCancel)


##############################################################################
#                          Método: populateTree                              #
#                                                                            #
# El cometido del presente método es el de rellenar el árbol en              #
# virtud de los valores y la lista de etiquetas, enviados como               #
# argumentos. El resultado será un árbol de dos columnas                     #
# (OID/Valor), donde la segunda columna únicamente tomará un valor           #
# visible para los nodos hoja. Este método logra mostrar la                  #
# información devuelta como resultado de un snmpwalk con un formato          #
# equivalente al que obtendríamos ejecutando un snmptranslate.               #
##############################################################################
def populateTree(self, valores, listaSnmp):

    # Limpiamos lo que pudiese existir en el árbol, establecemos el número de
    # columnas y sus etiquetas de cabecera y hacemos que los elementos nodo
    # de nuestro árbol sean expandibles.
    self.treeWidget.clear()
    self.treeWidget.setColumnCount(2)
    self.treeWidget.setHeaderLabels(["OID", "Valor"])
    self.treeWidget.setItemsExpandable(True)

    # Variables auxiliares para nuestro algoritmo.
    # top será usada para determinar el elemento padre de todos los nodos, mientras
    que
    # listaHijos será una lista donde iremos introduciendo todos los nodos (como
    elementos
    # QTreeWidgetItem) y listaHijosStr almacenará las etiquetas de todos los nodos
    # para llevar a cabo una función de control. Por su parte, i, será un contador
    # que nos permitirá llevar la cuenta del número de filas en listaSnmp.
    top = None
    self.listaHijos = []
    i = 0

    # Explicación del algoritmo
    # Cada "fila" presenta el siguiente formato (para el ejemplo utilizamos sysDescr
    ):
    # ('iso', 'org', 'dod', 'internet', 'mgmt', 'mib-2', 'system', 'sysDescr')
    # Por lo tanto, cada fila corresponde a las distintas etiquetas de un OID
    determinado.
    # Todos los elementos de la fila que no sean el último, son, seguro, un elemento
     nodo.
    # Es por ello que obtendremos la longitud de cada fila y, para cada elemento que
     no sea
    # el último (y por tanto, elemento nodo), se le asignará el valor "". El último
    elemento
    # de cada fila será quien haya de llevar el valor enviado en valores[k], con k =
     n"o de fila.
    # Para la primera fila, asignaremos el primer elemento como "top" y el resto le
    serán
    # añadidos recursivamente como hijos. Además, todos los elementos serán añadidos
     como
    # string a la lista listaHijosStr.
    # Para el resto de filas se sigue el mismo criterio con los valores,
    determinando si
    # ha de pintarse o no la etiqueta y su lugar en función de si existe o no en
    # listaHijosStr, y gracias al contador de control j, que nos indica en qué
    posición estamos.
    for fila in listaSnmp:
```

```
        aux = len(fila)

        # Primera fila
        if i == 0:
            j = 0
            for elem in fila:
                if top is None:
                    top = QTreeWidgetItem(self.treeWidget, [str(elem), ""])
                    self.listaHijos.append(top)
                    self.listaHijosStr.append(elem)

                else:
                    if j == (aux -1):
                        hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), str
(valores[i])])
                        hola.setFlags(hola.flags() | Qt.ItemIsEditable)
                        self.listaHijos.append(hola)
                        self.listaHijos.append(elem)
                    else:
                        hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), ""
])
                        self.listaHijos.append(hola)
                        self.listaHijosStr.append(elem)
                j += 1

        # Filas distintas a la primera
        else:
            j = 0
            for elem in fila:
                if j == (aux - 1):
                    hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), str(
valores[i])])
                    hola.setFlags(hola.flags() | Qt.ItemIsEditable)
                    self.listaHijos.append(hola)
                    self.listaHijos.append(elem)

                else:
                    if elem not in self.listaHijosStr:
                        hola = QTreeWidgetItem(self.listaHijos[j-1], [str(elem), ""
])
                        self.listaHijos.insert(j, hola)
                        self.listaHijosStr.insert(j, elem)
                j += 1
        i += 1


##########################################################################
#                       Método: recogerCambios                           #
#                                                                        #
# Este método tiene como objetivo recoger los cambios introducidos       #
# por el usuario en los elementos del árbol. Cuando un nodo cambia       #
# de valor se emite una señal Qt llamada "itemChanged", que              #
# conectamos al presente método. Ante este hecho, obtendremos el         #
# elemento alterado (elementoAlterado.text(0)) y el nuevo valor          #
# dado por el usuario (elementoAlterado.text(1)). Guardaremos esto       #
# en el parámetro publico "cambios", que será posteriormente             #
# accedido por MainWindow. Obtendremos el padre, el número de hijos      #
# y la posición de este elemento entre los hijos del padre, para         #
# comprobar si su nombre es único (en cuyo caso habrá que añadir un      #
# 0) o si su nombre es el mismo que el de sus hermanos (en cuyo          #
# caso se le añadirá el índice de hijo). Además, recorreremos            #
# listaHijos a fin de dibujar la ruta completa del OID alterado.         #
# Toda la ruta se devuelve como una lista. Para sysName presentaría      #
# el siguiente formato:                                                  #
# ('iso', 'org', 'dod', 'internet', 'mgmt', 'mib-2', 'system',           #
# 'sysName', '0'). Esta ruta será posteriormente accedida por            #
# MainWindow, que la reformateará de manera que sea interpretable        #
# por PySNMP. Por último, indicamos que se han producido cambios.        #
##########################################################################
def recogerCambios(self):
    elementoAlterado = self.treeWidget.currentItem()
    temporal = []
    temporal.append(elementoAlterado.text(0))
    temporal.append(elementoAlterado.text(1))
    self.cambios.append(temporal)

    padre = elementoAlterado.parent()
    ruta = []

    numHijos = padre.childCount()
    if numHijos > 1:
        if padre.indexOfChild(elementoAlterado) == 1:
            otroHijo = padre.child(2)
```

```
                else:
                    otroHijo = padre.child(1)
            if otroHijo:
                if otroHijo.text(0) == elementoAlterado.text(0):
                    completarOidCon = str(padre.indexOfChild(elementoAlterado))
                else:
                    completarOidCon = '0'

            yaEsta = False
            for item in self.listaHijos:
                if (item.text(0) != padre.text(0)) and not yaEsta:
                    ruta.append(item.text(0))
                elif item.text(0) == padre.text(0):
                    ruta.append(item.text(0))
                    yaEsta = True
                else:
                    break

            ruta.append(elementoAlterado.text(0))
            ruta.append(completarOidCon)
            self.cambiosRutas.append(ruta)

            if self.controlCambio is False:
                self.controlCambio = True


        ##########################################################################
        #                        Método: pulsaronOk                              #
        #                                                                        #
        # Tras la pulsación del botón "Ok" por parte del usuario, cerramos       #
        # la ventana a fin de que los valores cambiados sean interpretados       #
        # por MainWindow y las posibles peticiones realizadas a través del       #
        # controlador central por el SnmpServer.                                 #
        ##########################################################################
        def pulsaronOk(self):
            self.close()


        ##########################################################################
        #                      Método: pulsaronCancel                            #
        #                                                                        #
        # Ante la pulsación de "Cancel" por parte del usuario, cerramos la       #
        # ventana, dejando en False el control de cambios a fin de que las       #
        # posibles modificaciones que se hayan realizado no se apliquen en       #
        # el agente.                                                             #
        ##########################################################################
        def pulsaronCancel(self):
            self.controlCambio = False
            self.close()


# Hacemos que el diálogo SetWalkNext se ejecute si es llamado desde la función main(),
    presente en ventanaPrincipal.pyw
if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = SetWalkNextDlg()
    form.show()
    app.exec_()
```

Apéndice B

busquedas.py

```
#!/usr/bin/env python3.5
# -*- coding: utf-8 -*-
# Fichero: busquedas.py
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
###############################################################################
# Descripción: El objetivo del siguiente fichero es representar el            #
# diálogo encargado de preparar las búsquedas en la BBDD.                     #
###############################################################################

# Importaciones generales
import sys

# Importaciones de PyQT
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.QtSql import *

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
###############################################################################
#                          Historial de versiones                             #
# ----------------------------------------------------------------------------#
# Versión 1.0.0: 19 de marzo de 2018                                          #
#                  Versión inicial                                            #
# Versión 1.0.1: 1 de abril de 2018                                           #
#                  Modificaciones debidas al cambio de BBDD a SQLite.         #
# Versión 1.1.0: 10 de abril de 2018                                          #
#                  Primera versión totalmente operativa.                      #
###############################################################################

###############################################################################
#                               Clase: Buscar                                 #
#                                                                             #
# La clase Buscar(QDialog) tiene como objetivo generar un diálogo             #
# dinámico (de 1 a 8 campos de búsqueda), recogiendo los valores              #
# introducidos por el usuario y preparando la posterior búsqueda.             #
# El diálogo va a componerse de 3 ítems con disposición horizontal.           #
# Al primero de ellos lo llamaremos leftLayout, por ser el layout             #
# situado a la izquierda. El segundo será una línea vertical que irá          #
# creciendo a medida que se añadan más opciones al diálogo. El tercer         #
# ítem será la botonera.                                                      #
# El leftLayout, a su vez, será un layout vertical compuesto por los          #
# radio botones como primer elemento, y los distintos posibles campos         #
# a continuación. Los radiobotones permitirán elegir si se quiere una         #
# búsqueda como igualdad, desigualdad o contención y si, en el caso de        #
# múltiples campos, si se quiere una AND o una OR lógica. Los campos de       #
# búsqueda estarán compuestos por un desplegable para seleccionar el          #
# parámetro por el que buscar y por una línea de inserción de texto.          #
# La botonera presenta cuatro botones dispuestos de forma vertical, a         #
# saber, "Ok", "Cancelar", "+" y "-". Los botones "+" y "-" sirven            #
# para añadir o eliminar campos.                                              #
###############################################################################
```

```
class Buscar(QDialog):

    # Inicialización
    def __init__(self, parent=None):
        super(Buscar, self).__init__(parent)

        # Iniciamos las variables clave
        self.busqueda = QSqlQuery()
        self.inserciones = []
        self.comboBoxes = []
        self.moreFrames = []
        self.preparatoria = "SELECT * FROM elementos WHERE "
        listaOpciones = ["Dirección MAC", "Dirección IP", "Último descubrimiento", \
                         "communityRO", "communityRW"]
        self.moreFramesVisible = 0

        self.tituloAdvertencia = "Error en la búsqueda"

        # Left layout
        # GroupBox para los tres primeros radiobotones
        self.igual = QRadioButton("Igual")
        self.noIgual = QRadioButton("Distinto")
        self.contiene = QRadioButton("Contiene")
        unaDeTres = QHBoxLayout()
        unaDeTres.addWidget(self.igual)
        unaDeTres.addWidget(self.noIgual)
        unaDeTres.addWidget(self.contiene)
        equalNotContains = QGroupBox()
        equalNotContains.setLayout(unaDeTres)

        # GroupBox para los radiobotones AND/OR
        self.andLogica = QRadioButton("AND")
        self.orLogica = QRadioButton("OR")
        unaDeDos = QHBoxLayout()
        unaDeDos.addWidget(self.andLogica)
        unaDeDos.addWidget(self.orLogica)
        andOr = QGroupBox()
        andOr.setLayout(unaDeDos)

        # Creamos el layout horizontal de radiobotones
        radioBotones = QHBoxLayout()
        radioBotones.addWidget(equalNotContains)
        radioBotones.addWidget(andOr)

        # Generamos 8 desplegables (con las opciones definidas en listaOpciones) y 8
    campos de inserción
        i = 0
        while i<8:
            listaOpcionesBox = QComboBox()
            listaOpcionesBox.addItems(listaOpciones)
            self.comboBoxes.insert(i, listaOpcionesBox)

            lineaBusqueda = QLineEdit()
            lineaBusqueda.setMinimumSize(250, 1)
            self.inserciones.insert(i, lineaBusqueda)

            i += 1

        # Creamos un layout horizontal para almacenar el campo mostrado inicialmente.
        mostradoFijo = QHBoxLayout()
        mostradoFijo.addWidget(self.comboBoxes[0])
        mostradoFijo.addWidget(self.inserciones[0])

        # Generamos 7 QFrames con el resto de campos. Serán mostradas o no en virtud de
    la voluntad del usuario.
        self.moreFrame = QFrame()
        frameLayout = QHBoxLayout()
        frameLayout.addWidget(self.comboBoxes[1])
        frameLayout.addWidget(self.inserciones[1])
        self.moreFrame.setLayout(frameLayout)

        self.moreFrame2 = QFrame()
        frameLayout2 = QHBoxLayout()
        frameLayout2.addWidget(self.comboBoxes[2])
        frameLayout2.addWidget(self.inserciones[2])
        self.moreFrame2.setLayout(frameLayout2)

        self.moreFrame3 = QFrame()
        frameLayout3 = QHBoxLayout()
        frameLayout3.addWidget(self.comboBoxes[3])
        frameLayout3.addWidget(self.inserciones[3])
        self.moreFrame3.setLayout(frameLayout3)
```

```
        self.moreFrame4 = QFrame()
        frameLayout4 = QHBoxLayout()
        frameLayout4.addWidget(self.comboBoxes[4])
        frameLayout4.addWidget(self.inserciones[4])
        self.moreFrame4.setLayout(frameLayout4)

        self.moreFrame5 = QFrame()
        frameLayout5 = QHBoxLayout()
        frameLayout5.addWidget(self.comboBoxes[5])
        frameLayout5.addWidget(self.inserciones[5])
        self.moreFrame5.setLayout(frameLayout5)

        self.moreFrame6 = QFrame()
        frameLayout6 = QHBoxLayout()
        frameLayout6.addWidget(self.comboBoxes[6])
        frameLayout6.addWidget(self.inserciones[6])
        self.moreFrame6.setLayout(frameLayout6)

        self.moreFrame7 = QFrame()
        frameLayout7 = QHBoxLayout()
        frameLayout7.addWidget(self.comboBoxes[7])
        frameLayout7.addWidget(self.inserciones[7])
        self.moreFrame7.setLayout(frameLayout7)

        # Creamos el leftLayout, con la parte fija y las moreFrame
        leftLayout = QVBoxLayout()
        leftLayout.addLayout(radioBotones)
        leftLayout.addLayout(mostradoFijo)
        leftLayout.addWidget(self.moreFrame)
        leftLayout.addWidget(self.moreFrame2)
        leftLayout.addWidget(self.moreFrame3)
        leftLayout.addWidget(self.moreFrame4)
        leftLayout.addWidget(self.moreFrame5)
        leftLayout.addWidget(self.moreFrame6)
        leftLayout.addWidget(self.moreFrame7)


        # Layout de la botonería
        botonOk = QPushButton("&OK")
        botonCerrar = QPushButton("&Cerrar")
        botonMas = QPushButton("+")
        botonMenos = QPushButton("-")

        buttonLayout = QVBoxLayout()
        buttonLayout.addWidget(botonCerrar)
        buttonLayout.addWidget(botonOk)
        buttonLayout.addWidget(botonMas)
        buttonLayout.addWidget(botonMenos)

        self.setWindowTitle("Búsquedas")

        # Línea para separar los layouts
        line = QFrame()
        line.setFrameStyle(QFrame.VLine | QFrame.Sunken)

        # Creamos el layout principal, compuesto por todos los elementos que hemos ido
    creando
        mainLayout = QHBoxLayout()
        mainLayout.addLayout(leftLayout)
        mainLayout.addWidget(line)
        mainLayout.addLayout(buttonLayout)
        self.setLayout(mainLayout)

        # "Escondemos" los 7 campos restantes, que serán mostrados si el usuario así lo
    quiere
        self.moreFrame.hide()
        self.moreFrame2.hide()
        self.moreFrame3.hide()
        self.moreFrame4.hide()
        self.moreFrame5.hide()
        self.moreFrame6.hide()
        self.moreFrame7.hide()

        mainLayout.setSizeConstraint(QLayout.SetFixedSize)

        # Realizamos las conexiones de la botonera
        self.connect(botonMas, SIGNAL("clicked()"), self.addField)
        self.connect(botonMenos, SIGNAL("clicked()"), self.deleteField)
        self.connect(botonOk, SIGNAL("clicked()"), self, SLOT("accept()"))
        self.connect(botonCerrar, SIGNAL("clicked()"), self, SLOT("reject()"))


##########################################################################
```

```
#                          Método: addField                                  #
#                                                                            #
# Este método es lanzado cuando el usuario pulsa sobre el botón "+"         #
# del presente diálogo. Se apoya en el parámetro moreFramesVisible,          #
# al que actualiza si fuera el caso, para determinar qué trama debe          #
# mostrar y para, llegado al límite, señalar al usuario que no es            #
# posible añadir más parámetros a la búsqueda.                               #
##############################################################################
def addField(self):

    if self.moreFramesVisible == 0:
        self.moreFrame.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 1:
        self.moreFrame2.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 2:
        self.moreFrame3.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 3:
        self.moreFrame4.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 4:
        self.moreFrame5.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 5:
        self.moreFrame6.show()
        self.moreFramesVisible += 1
    elif self.moreFramesVisible == 6:
        self.moreFrame7.show()
        self.moreFramesVisible += 1
    else:
        QMessageBox.warning(None, self.tituloAdvertencia, "El máximo de parámetros
 para la búsqueda es de 8")


##############################################################################
#                          Método: deleteField                               #
#                                                                            #
# Este método es lanzado cuando el usuario pulsa sobre el botón "-"          #
# del presente diálogo. Se apoya en el parámetro moreFramesVisible,          #
# al que actualiza si fuera el caso, para determinar qué trama debe          #
# esconder y para, llegado al límite, señalar al usuario que no es           #
# posible realizar una búsqueda sin parámetros.                              #
##############################################################################
def deleteField(self):

    if self.moreFramesVisible == 7:
        self.moreFrame7.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 6:
        self.moreFrame6.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 5:
        self.moreFrame5.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 4:
        self.moreFrame4.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 3:
        self.moreFrame3.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 2:
        self.moreFrame2.hide()
        self.moreFramesVisible -= 1
    elif self.moreFramesVisible == 1:
        self.moreFrame.hide()
        self.moreFramesVisible -= 1
    else:
        QMessageBox.warning(None, self.tituloAdvertencia, "No se puede realizar una
 búsqueda sin parámetros")


##############################################################################
#                          Método: accept                                    #
#                                                                            #
# Este método es invocado cuando el usuario pulsa en el botón "Ok".          #
# Su cometido es extraer toda la información que el usuario haya             #
# introducido, para que la consulta a la BBDD sea preparada. Así             #
# mismo, si el usuario no hubiese completado la selección de los             #
# radiobotones, sería avisado con un mensaje de advertencia y no se          #
# seguiría adelante con la búsqueda.                                         #
##############################################################################
```

```
def accept(self):

    # Comprobaciones de los radiobotones
    if not (self.andLogica.isChecked() or self.orLogica.isChecked()):
        QMessageBox.warning(None, self.tituloAdvertencia, "Debe seleccionar si desea
 búsqueda como AND o como OR.")
        return
    if not (self.igual.isChecked() or self.noIgual.isChecked() or self.contiene.
isChecked()):
        QMessageBox.warning(None, self.tituloAdvertencia, "Debe seleccionar el tipo
de búsqueda.")
        return

    # Extraemos la información de los radiobotones y la convertimos a notación SQL
    if self.igual.isChecked():
        simbolo = "="
    elif self.contiene.isChecked():
        simbolo = "LIKE"
    else:
        simbolo = "!="

    if self.andLogica.isChecked():
        logica = "AND"
    else:
        logica = "OR"

    # Llamamos al método buscarValores, con el símbolo y la lógica previamente extra
ídos
    self.buscarValores(simbolo, logica)

    # Ejecutamos la búsqueda con la variable "preparatoria" lista como petición SQL
 gracias
    # al método buscarValores
    self.busqueda.exec_(self.preparatoria)

    # Creamos una instancia de la clase RepresentarResultadoBuscar, que mostrará el
 resultado en forma
    # de tabla en una ventana emergente, pasándole la petición realizada, a fin de
que se extraigan
    # los resultados de esta y se representen de un modo "user-friendly".
    miPrueba = RepresentarResultadoBuscar(self.busqueda)
    miPrueba.exec_()

    # Una vez ejecutada la consulta, dejamos lista la variable preparatoria para
futuras búsquedas.
    self.preparatoria = "SELECT * FROM elementos WHERE "


##############################################################################
#                          Método: buscarValores                             #
#                                                                            #
# El presente método prepara la consulta a la base de datos.                 #
# Tomando como entradas el símbolo y la lógica en notación MySQL,            #
# recorre cada campo de inserción de la lista de QLineEdit y, si el          #
# valor en ella no es nulo, relaciona éste con el valor del                  #
# desplegable y lo asocia a la columna adecuada de la tabla                  #
# "elementos" de la base de datos.                                           #
##############################################################################
def buscarValores(self, simbolo, logica):

    # Inicializaciones. "i" servirá para ir repasando las posiciones de la lista de
 QLineEdit,
    # "campos" nos servirá para introducir el operador lógico en el caso de que el
usuario haya
    # seleccionado más de 1 campo y la lista "índices" nos relacionará el valor que
podemos
    # obtener de las comboBoxes con la columna correspondiente de la tabla "
elementos".
    i = 0
    campos = 0
    indices = ["mac_address", "ip_address", "lastTimeDetected", "communityRO", "
communityRW"]

    # Realizamos el bucle y preparamos la consulta
    for caja in self.comboBoxes:
        if self.inserciones[i].text() is not ' ' :
            if campos != 0:
                self.preparatoria = self.preparatoria + logica + " "
            if simbolo != "LIKE":
                self.preparatoria = self.preparatoria + indices[caja.currentIndex()]
 + " " + simbolo + " '" + \
                    self.inserciones[i].text() + "' "
            elif simbolo == "LIKE":
```

```
                    self.preparatoria = self.preparatoria + indices[caja.currentIndex()]
 + " " + simbolo + " '%" + \
                        self.inserciones[i].text() + "%' "

                campos += 1
            i += 1

        # Le introducimos a la consulta, una vez finalizado el bucle, el ';' final
    necesario.
        self.preparatoria = self.preparatoria + ";"


##########################################################################
#                    Clase: RepresentarResultadoBuscar                   #
#                                                                        #
# La presente clase genera un diálogo con una tabla dentro de él, de     #
# idéntico formato a la presente en el centralWidget de la ventana       #
# principal, para mostrar el resultado de la búsqueda realizada.         #
##########################################################################
class RepresentarResultadoBuscar(QDialog):

    # Inicialización
    def __init__(self, query, parent=None):
        super(RepresentarResultadoBuscar, self).__init__(parent)

        # Establecemos la tabla y sus propiedades
        self.tablaResultados = QTableWidget()
        self.tablaResultados.setColumnCount(5)
        self.tablaResultados.setHorizontalHeaderLabels(["Dir. MAC", "Dir. IP", "Último
 contacto", \
                                                        "CommunityRO", "CommunityRW"])
        self.tablaResultados.setAlternatingRowColors(True)
        self.tablaResultados.setEditTriggers(QTableWidget.NoEditTriggers)
        self.tablaResultados.setSelectionBehavior(QTableWidget.SelectRows)
        self.tablaResultados.setSelectionMode(QTableWidget.SingleSelection)
        self.setWindowTitle("Resultado de la búsqueda")

        # Llamamos al método actualizarTabla con la consulta SQL realizada y pasada como
     parámetro
        self.actualizarTabla(query)


    ##########################################################################
    #                      Método: actualizarTabla                           #
    #                                                                        #
    # Este método se encarga de, pasado el resultado de una consulta,       #
    # representarlo con el formato adecuado. Recorre para ello las           #
    # distintas filas devueltas por la BBDD y va rellenando con ello la      #
    # tabla. Se vale del tamaño de la respuesta para establecer el           #
    # número de filas a representar.                                         #
    ##########################################################################
    def actualizarTabla(self, query):

        # Obtenemos el número de filas de la respuesta dada por la BBDD y fijamos
        # con ello el número de filas de la QTableWidget.
        numFilas = 0
        while query.next():
            numFilas += 1
        self.tablaResultados.setRowCount(numFilas)

        # Como estábamos al final de la query tras el bucle que determina el número de
     filas,
        # nos colocamos al principio para el bucle que rellena esas filas.
        # Posteriormente recorremos el resultado de la búsqueda para rellenar las celdas
     de la GUI.
        # Como podemos encontrarnos ante valores nulos en los campos de las Communities,
     debemos
        # manejar las posibles excepciones que esto pueda arrojarnos.
        query.seek(-1)
        i = 0
        while query.next():
            mac_address = QTableWidgetItem(query.value(0))
            ip_address = QTableWidgetItem(query.value(1))
            lastTimeDetected = QTableWidgetItem(query.value(2))

            try:
                if query.value(3).isNull():
                    communityRO = QTableWidgetItem(' ')
            except:
                communityRO = QTableWidgetItem(query.value(3))

            try:
                if query.value(4).isNull():
```

```
                communityRW = QTableWidgetItem(' ')
            except:
                communityRW = QTableWidgetItem(query.value(4))

            self.tablaResultados.setItem(i, 0, mac_address)
            self.tablaResultados.setItem(i, 1, ip_address)
            self.tablaResultados.setItem(i, 2, lastTimeDetected)
            self.tablaResultados.setItem(i, 3, communityRO)
            self.tablaResultados.setItem(i, 4, communityRW)

            i += 1

        self.tablaResultados.resizeColumnsToContents()
        self.tablaResultados.resizeRowsToContents()
        self.muestra = QVBoxLayout()
        self.muestra.addWidget(self.tablaResultados)

        self.setLayout(self.muestra)


# Hacemos que el diálogo Buscar se ejecute si es llamado desde la función main(),
    presente en ventanaPrincipal.pyw
if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Buscar()
    form.show()
    app.exec_()
```

Apéndice C

controlador.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Fichero: controlador.py
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
##############################################################################
# Descripción: El siguiente es un fichero fundamental para la                #
# aplicación. Será el intermediario entre los distintos elementos            #
# de la aplicación.                                                          #
##############################################################################

# Importaciones generales
import sys
import datetime

# Importaciones de PySNMP
from pysnmp import smi, carrier

# Importaciones de PyQt
from PyQt4.QtGui import *
from PyQt4.QtSql import *

# Para leer la tabla ARP
from python_arptable import get_arp_table

# Importaciones para lectura/escritura de config.xml
# cElemenTree es mucho más rápida y consume menos memoria que elemenTree
try:
    import xml.etree.cElementTree as ET
except importError:
    import xml.etree.ElementTree as ET

# Importaciones de nuestro proyecto
import snmpserver

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
##############################################################################
#                          Historial de versiones                            #
# ------------------------------------------------------------------------   #
# Versión 1.0.0: 6 de marzo de 2018                                          #
#                  Versión inicial                                           #
# Versión 1.0.1: 9 de marzo de 2018                                          #
#                  Cambio de máscara de subred a prefijo de subred           #
# Versión 1.0.2: 13 de marzo de 2018                                         #
#                  Inicialización de mibBuilder.                             #
#                  Desarrollo del método loadMib.                            #
#                  Desarrollo del método loadCompiler.                       #
#                  Carga inicial de módulos MIB.                             #
# Versión 1.0.3: 15 de marzo de 2018                                         #
#                  Apertura de conexión con la base de datos.                #
#                  Lectura/escritura en la base de datos para resync()       #
# Versión 1.0.4: 22 de marzo de 2018                                         #
```

```
#                    Desarrollo del método peticiones.                        #
# Versión 1.0.5: 1 de abril de 2018                                           #
#                    Modificaciones debidas al cambio de BBDD a SQLite.       #
# Versión 1.1.0: 10 de abril de 2018                                          #
#                    Primera versión totalmente operativa.                    #
################################################################################

################################################################################
#                         Clase: ControladorCentral                            #
#                                                                              #
# Esta clase proporciona los métodos y atributos necesarios para llevar        #
# a cabo la labor de controlador central. Junto a la clase SnmpServer,         #
# presente en el fichero snmpserver.py, constituye el controlador en           #
# virtud de la programación Modelo/Vista/Controlador.                          #
################################################################################
class ControladorCentral():

    # Inicialización
    def __init__(self):
        super(ControladorCentral, self).__init__()

        # Nos creamos una instancia de SnmpServer como parte del controlador
        self.miServidor = snmpserver.SnmpServer()

        # Establecemos el nombre de nuestra base de datos
        nombrebbdd = "valladomib.db"

        # Cargamos e intentamos abrir la base de datos, operando con SQLite
        db = QSqlDatabase.addDatabase("QSQLITE")
        db.setDatabaseName(nombrebbdd)
        if not db.open():
            QMessageBox.warning(None, "Error de base de datos", db.lastError().text())
            sys.exit(1)


    ############################################################################
    #                            Método: resync                                #
    #                                                                          #
    # Este método se encargará de realizar un escaneo en la porción de        #
    # red indicada en config.xml. Para ello, se valdrá del método             #
    # snmpget, de la clase SnmpServer, preguntando en cada dirección           #
    # presente en el rango por el sysObjectID. Una vez descubiertos,           #
    # obtendrá algún dato adicional (dirección MAC y otros parámetros          #
    # vía SNMP) y escribirá en la base de datos, para la que se ha             #
    # abierto una conexión previamente. Este método es llamado desde           #
    # la ventana principal (clase MainWindow), a petición del usuario          #
    # cuando pulse sobre el icono correspondiente.                             #
    ############################################################################
    def resync(self):

        # Elegimos nuestro fichero de configuración
        tree = ET.parse('config.xml')

        # Extraemos de dicho fichero dirección, la máscara de subred y la community de
     lectura
        for elem in tree.iter():
            if elem.tag == 'Subnet':
                subnetIpAddress = elem.text
            elif elem.tag == 'Prefix':
                self.subnetPrefix = int(elem.text)
            elif elem.tag == 'CommunityRO':
                communityRO = elem.text

        # Creamos las variables necesarias para el parseo de estas direcciones
        subnetIpByte1=''
        subnetIpByte2=''
        subnetIpByte3=''
        subnetIpByte4=''
        primerByteSI = False
        segundoByteSI = False
        tercerByteSI = False

        # Dividimos la dirección proporcionada en bytes
        for char in subnetIpAddress:
            if char is not '.' and primerByteSI is not True:
                subnetIpByte1 = subnetIpByte1 + char
            elif char == '.' and primerByteSI is not True:
                primerByteSI = True
            elif char is not '.' and segundoByteSI is not True:
                subnetIpByte2 = subnetIpByte2 + char
            elif char == '.' and segundoByteSI is not True:
                segundoByteSI = True
            elif char is not '.' and tercerByteSI is not True:
```

```
            subnetIpByte3 = subnetIpByte3 + char
        elif char == '.' and tercerByteSI is not True:
            tercerByteSI = True
        else:
            subnetIpByte4 = subnetIpByte4 + char

    # Hacemos que el resultado obtenido sea tratado como un entero
    subnetIpByte1 = int(subnetIpByte1)
    subnetIpByte2 = int(subnetIpByte2)
    subnetIpByte3 = int(subnetIpByte3)
    subnetIpByte4 = int(subnetIpByte4)

    # Transformamos el prefijo leído del XML a bytes de la máscara de subred
    if self.subnetPrefix > 8:
        subnetMaskByte1 = 255
        if self.subnetPrefix > 16:
            subnetMaskByte2 = 255
            if self.subnetPrefix > 24:
                subnetMaskByte3 = 255
                subnetMaskByte4 = self.ayudaMask(self.subnetPrefix - 24)
            else:
                subnetMaskByte3 = self.ayudaMask(self.subnetPrefix - 16)
                subnetMaskByte4 = 0
        else:
            subnetMaskByte2 = self.ayudaMask(self.subnetPrefix - 8)
            subnetMaskByte3 = 0
            subnetMaskByte4 = 0
    else:
        subnetMaskByte1 = self.ayudaMask(self.subnetPrefix)
        subnetMaskByte2 = 0
        subnetMaskByte3 = 0
        subnetMaskByte4 = 0

    # Preparamos las variables necesarias para obtener nuestro primer
    # y nuestro último ítem.
    subnetByte1 = subnetMaskByte1 & subnetIpByte1
    subnetByte2 = subnetMaskByte2 & subnetIpByte2
    subnetByte3 = subnetMaskByte3 & subnetIpByte3
    subnetByte4 = subnetMaskByte4 & subnetIpByte4
    primerByte = False
    segundoByte = False
    tercerByte = False
    primero = []
    ultimo = []

    # En función de la máscara de subred, encontramos dónde terminan los bits
    # a "1" y extraemos las direcciones de subred y broadcast. El primer
    # elemento a evaluar será el siguiente a la dirección de subred y el
    # último será el previo a la dirección de broadcast.
    if subnetMaskByte1 == 255:
        primero.append(subnetByte1)
        ultimo.append(subnetByte1)
        if subnetMaskByte2 == 255:
            primero.append(subnetByte2)
            ultimo.append(subnetByte2)
            if subnetMaskByte3 == 255:
                primero.append(subnetByte3)
                ultimo.append(subnetByte3)
                # Seleccionamos como primero la dirección siguiente a la de subred
                primero.append(subnetByte4 + 1)
                # Seleccionamos como último la dirección previa a la de broadcast
                ultimoItemByte4 = (((1 << 8) - 1 - subnetMaskByte4) | subnetIpByte4)
- 1
                ultimo.append(ultimoItemByte4)
            else:
                primero.append(subnetByte3)
                primero.append(subnetByte4 + 1)
                ultimoItemByte3 = (((1 << 8) - 1 - subnetMaskByte3) | subnetIpByte3)
                ultimo.append(ultimoItemByte3)
                ultimo.append(254)
                tercerByte = True
        else:
            primero.append(subnetByte2)
            primero.append(subnetByte3)
            primero.append(subnetByte4 + 1)
            ultimoItemByte2 = (((1 << 8) - 1 - subnetMaskByte2) | subnetIpByte2)
            ultimo.append(ultimoItemByte2)
            ultimo.append(255)
            ultimo.append(254)
            segundoByte = True
    else:
        primero.append(subnetByte1)
        primero.append(subnetByte2)
```

```
            primero.append(subnetByte3)
            primero.append(subnetByte4 + 1)
            ultimoItemByte1 = (((1 << 8) - 1 - subnetMaskByte1) | subnetIpByte1)
            ultimo.append(ultimoItemByte1)
            ultimo.append(255)
            ultimo.append(255)
            ultimo.append(254)
            primerByte = True

        # Agruparemos los elementos descubiertos como un diccionario, con clave su dirección IP
        # y valor su sysObjectID.
        elementos = {}

        # Este método genera una pérdida de 5 segundos por cada hueco vacío (por el timeout ARP)
        # De momento se encuentra únicamente desarrollado para subredes de clase C o menores
        # Realizamos un bucle entre el primer y el último elemento del rango a inspeccionar,
        # enviando un snmpget del sysObjectID para cada elemento presente en dicho rango.
        # Si obtenemos respuesta, guardamos la parte de la respuesta que se encuentre
        # tras el '=' (pues viene en formato SNMPv2-MIB::system-sysObjectID = valor),
        # damos formato adecuado para su IPv4 y lo almacenamos en nuestro diccionario de elementos

        # Máscara entre 0 y 7 bits
        if primerByte:
            confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar una resincronización para"
                                                              " 33.554.430 o más direcciones. ¿Seguro que quiere continuar?"
                                                              " Puede modificar la configuración si lo desea antes de"
                                                              " realizar esta resincronización. Pulsar 'OK' continuaría"
                                                              " con la resincronización.",
                                         QMessageBox.Ok, QMessageBox.Cancel)
            if confir == QMessageBox.Ok:
                pb = primero[0]
                sb = primero[1]
                tb = primero[2]
                cb = primero[3]
                ul = 255
                while pb <= ultimo[0]:
                    while sb <= ultimo[1]:
                        while tb <= ultimo[2]:
                            if tb == ultimo[2]:
                                ul = 254
                            while cb <= ul:
                                posibleElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(tb) + '.' + str(cb)
                                print("Inspeccionando la dirección " + posibleElemento)
                                respuestaDiscover = self.miServidor.snmpget(".1.3.6.1.2.1.1.2.0", posibleElemento, communityRO)
                                cb += 1
                                if respuestaDiscover is not None:
                                    respuestaDiscover = str(respuestaDiscover)
                                    ipElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(tb) + '.' + str(cb - 1)
                                    elementos[ipElemento] = respuestaDiscover
                            tb += 1
                        sb += 1
                    pb += 1
            else:
                return

        # Máscara entre 8 y 15 bits
        elif segundoByte:
            confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar una resincronización para"
                                                              " 131.070 o más direcciones. ¿Seguro que quiere continuar?"
                                                              " Puede modificar la configuración si lo desea antes de"
                                                              " realizar esta resincronización. Pulsar 'OK' continuaría"
                                                              " con la resincronización.",
                                         QMessageBox.Ok, QMessageBox.Cancel)
            if confir == QMessageBox.Ok:
                sb = primero[1]
                tb = primero[2]
```

```
            cb = primero[3]
            ul = 255
            while sb <= ultimo[1]:
                while tb <= ultimo[2]:
                    if tb == ultimo[2]:
                        ul = 254
                    while cb <= ul:
                        posibleElemento = str(primero[0]) + '.' + str(primero[1]) +
'.' + str(tb) + '.' + str(cb)
                        print("Inspeccionando la dirección " + posibleElemento)
                        respuestaDiscover = self.miServidor.snmpget("
.1.3.6.1.2.1.1.2.0", posibleElemento,
                                                                    communityRO)
                        cb += 1
                        if respuestaDiscover is not None:
                            respuestaDiscover = str(respuestaDiscover)
                            ipElemento = str(primero[0]) + '.' + str(primero[1]) + '
.' + str(tb) + '.' + str(cb - 1)
                            elementos[ipElemento] = respuestaDiscover
                    tb += 1
                sb += 1
        else:
            return

    # Máscara entre 16 y 23 bits
    elif tercerByte:
        confir = QMessageBox.warning(None, "Advertencia", "Está a punto de realizar
una resincronización para"
                                                          " 510 o más direcciones. ¿Seguro que
 quiere continuar?"
                                                          " Puede modificar la configuración
si lo desea antes de"
                                                          " realizar esta resincronización.
Pulsar 'OK' continuaría"
                                                          " con la resincronización.",
QMessageBox.Ok, QMessageBox.Cancel)
        if confir == QMessageBox.Ok:
            tb = primero[2]
            cb = primero[3]
            ul = 255
            while tb <= ultimo[2]:
                if tb == ultimo[2]:
                    ul = 254
                while cb <= ul:
                    posibleElemento = str(primero[0]) + '.' + str(primero[1]) + '.'
+ str(tb) + '.' + str(cb)
                    print("Inspeccionando la dirección " + posibleElemento)
                    respuestaDiscover = self.miServidor.snmpget(".1.3.6.1.2.1.1.2.0"
, posibleElemento, communityRO)
                    cb += 1
                    if respuestaDiscover is not None:
                        respuestaDiscover = str(respuestaDiscover)
                        ipElemento = str(primero[0]) + '.' + str(primero[1]) + '.' +
 str(tb) + '.' + str(cb-1)
                        elementos[ipElemento] = respuestaDiscover
                tb += 1
                cb = 0
        else:
            return

    # Máscara de 24 o más bits
    else:
        i = primero[3]
        while i <= ultimo[3]:
            posibleElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(
primero[2]) + '.' + str(i)
            print("Inspeccionando la dirección " + posibleElemento)
            respuestaDiscover = self.miServidor.snmpget(".1.3.6.1.2.1.1.2.0",
posibleElemento, communityRO)
            i = i+1
            if respuestaDiscover is not None:
                respuestaDiscover = str(respuestaDiscover)
                ipElemento = str(primero[0]) + '.' + str(primero[1]) + '.' + str(
primero[2]) + '.' + str(i-1)
                elementos[ipElemento] = respuestaDiscover

    # Hacemos el walk para los elementos descubiertos y preparamos la consulta a la
BBDD
    for elemento in elementos:
        diccionarioQuery = {"mac_address":'', "ip_address":elemento, "
lastTimeDetected":datetime.datetime.today()}

        # Realizamos el snmpget que antes nos dio buenos resultados para actualizar
```

```
la caché ARP
        self.miServidor.snmpget(".1.3.6.1.2.1.1.2.0", elemento, communityRO)

        # Obtenemos la tabla ARP para, relacionar la dirección MAC con la IP del
elemento descubierto
        # y la consultamos para enviarla en la consulta SQL.
        tablaArp = get_arp_table()
        for entrada in tablaArp:
            if entrada['IP address'] == elemento:
                diccionarioQuery["mac_address"] = entrada['HW address']

        # Creamos una instancia QSqlQuery para realizar nuestra petición
        petiSql = QSqlQuery()

        # Preparamos una actualización si la MAC del elemento ya estaba en la BBDD.
        # Podría darse un error de MAC duplicada que no captáramos, pero, dado que
su aparición sucede
        # con poca frecuencia, lo dejamos fuera del alcance de nuestra aplicación,
siendo el usuario el que
        # deberá proporcionar los adecuados métodos para evitar este problema.
        peticion1 = "UPDATE elementos SET ip_address='" + diccionarioQuery["
ip_address"] +\
                    "', lastTimeDetected='" + str(diccionarioQuery["lastTimeDetected
"]) +\
                    "' WHERE mac_address='"+ diccionarioQuery["mac_address"] + "';"

        # Ejecutamos la petición previamente preparada
        petiSql.exec_(peticion1)

        # Si no existía ningún elemento con esa MAC, no habremos escrito nada en la
BBDD y, por lo tanto,
        # preparamos la consulta de inserción con los datos presentes en nuestro
diccionario.
        if petiSql.numRowsAffected() == 0:
            peticion2 = "INSERT INTO elementos (mac_address, ip_address,
lastTimeDetected) VALUES ('"\
                        + diccionarioQuery["mac_address"] + "', '" + \
            diccionarioQuery["ip_address"] + "', '" + str(diccionarioQuery["
lastTimeDetected"]) + "');"

            # Ejecutamos la petición previamente preparada
            try:
                petiSql.exec_(peticion2)
            except:
                QMessageBox.warning(None, "Error SQL", petiSql.lastError().text())


##########################################################################
#                          Método: ayudaMask                             #
#                                                                        #
# El cometido de este método es devolver el valor de un byte de la       #
# máscara de subred, proporcionado el número de bits a 1 en dicho        #
# octeto. Es llamado desde el método resync de la misma clase.           #
##########################################################################
def ayudaMask(self, num):
    salidaMask = 0
    auxiliar = num
    auxiliar2 = 7
    while auxiliar > 0:
        salidaMask = salidaMask + 2 ** auxiliar2
        auxiliar -= 1
        auxiliar2 -= 1
    return salidaMask


##########################################################################
#                          Método: loadMib                               #
#                                                                        #
# Método destinado a ejercer la conexión entre la llamada desde          #
# LoadMibDlgModule.accept() y el método loadMib de SnmpServer.           #
##########################################################################
def loadMib(self, *args):
    return self.miServidor.loadMib(*args)


##########################################################################
#                          Método: loadCompiler                          #
#                                                                        #
# Método destinado a ejercer la conexión entre la llamada desde          #
# LoadMibDlgCompiler.accept() y el método loadCompiler de                #
# SnmpServer.                                                            #
##########################################################################
def loadCompiler(self, arg):
```

```
        self.miServidor.loadCompiler(arg)

    ##########################################################################
    #                            Método: peticiones                          #
    #                                                                        #
    # Método encargado de trasladar las peticiones realizadas desde la       #
    # interfaz gráfica al servidor y parsear las respuestas para             #
    # entregar el formato adecuado a la GUI. Recibe como argumentos los      #
    # parámetros consulta (get, getNext, walk o set), argumento (OID o       #
    # MIB objeto de la operación), direccion (IP del elemento) y, si         #
    # procediera, nuevoValor (para snmpset). Lee del fichero config.xml      #
    # las communities necesarias (o de la BBDD, si el elemento tuviese       #
    # communities específicas) y envía la petición, con su community,        #
    # al método de SnmpServer adecuado, procesando, si procediese, sus       #
    # respuestas. Se devuelve a la GUI el resultado apropiado.               #
    ##########################################################################
    def peticiones(self, consulta, argumento, direccion, *nuevoValor):

        # Llamamos al método determinarComm para hallar las communities RO y RW
        communityro = self.determinarComm(direccion, 'communityRO')
        communityrw = self.determinarComm(direccion, 'communityRW')

        if communityro == ''  or communityrw == '' or communityro == None or
    communityrw == None:
            QMessageBox.warning(None, "Error de configuración", "Ha ocurrido un error en
     la lectura de la configuración.")
            return

        if consulta == "get":
            self.miServidor.snmpget(argumento, direccion, communityro)

        elif consulta == "getnext":
            try:
                valores, etiquetas = self.miServidor.snmpwalk(direccion, communityro,
    False, argumento)
            except smi.error.MibNotFoundError:
                QMessageBox.warning(None, "Error", "Su petición no ha podido ser
    atendida. El módulo"
                                                   " que ha introducido no se reconoce
    como un módulo válido."
                                                   " Puede revisar su inserción o bien
    cargar el módulo"
                                                   " que desee.")
                return
            return valores, etiquetas

        elif consulta == "walk":
            try:
                valores, etiquetas = self.miServidor.snmpwalk(direccion, communityro,
    True)
            except TypeError:
                QMessageBox.warning(None, "Error", "Su petición no ha podido ser
    atendida. Es posible"
                                                   "que no tenga comunicación con el
    agente seleccionado.")
                return
            return valores, etiquetas

        elif consulta == "set":
            self.miServidor.snmpset(argumento, direccion, communityrw, nuevoValor[0])

        else:
            QMessageBox.warning(None, "Error de consulta SNMP", "La consulta indicada es
     errónea. Por favor, revísela y vuelva a intentarlo.")


    ##########################################################################
    #                            Método: wakeUp                              #
    #                                                                        #
    # Método invocado al arranque de la aplicación. Tras leer de la          #
    # base de datos los elementos gestionados se quiere, para su             #
    # representación, obtener los parámetros relativos a                     #
    # SNMPv2-MIB::system. Así, además, podremos determinar si existe o       #
    # no conectividad con el elemento en cuestión. Así, primero obten-       #
    # dremos la communityRO, para tratar luego de realizar un walk del       #
    # módulo SNMPv2-MIB. Si tiene éxito, generamos una marca horaria y       #
    # devolvemos los resultados solicitados.                                 #
    ##########################################################################
    def wakeUp(self, ipAddr):

        comm = self.determinarComm(ipAddr, 'communityRO')
```

```
    try:
        valores, etiquetas = self.miServidor.snmpwalk(ipAddr, comm, False, 'SNMPv2-
MIB')

        i = 0
        for elem in valores:
            if i == 0:
                sysDescr = elem
            elif i == 1:
                sysObjectID = elem
            elif i == 2:
                sysUpTime = elem
            elif i == 3:
                sysContact = elem
            elif i == 4:
                sysName = elem
            elif i == 5:
                sysLocation = elem
            elif i == 6:
                sysServices = elem
            i += 1
        lastTimeDetected = datetime.datetime.today()
    except:
        return

    return str(lastTimeDetected), sysDescr, sysObjectID, sysUpTime, sysContact,
sysName, sysLocation, sysServices


##########################################################################
#                    Método: determinarComm                              #
#                                                                        #
# El presente método se encarga, dada la IP de un elemento y qué         #
# community se quiere, determinar cuál es la community a emplear.        #
# Realiza una búsqueda en la base de datos. Si el elemento tiene         #
# aquí especificada la community, esta es empleada. En caso              #
# contario, extrae la community de las opciones generales                #
# especificadas en config.xml                                            #
##########################################################################
def determinarComm(self, ip, rorw):

    buscaComm = QSqlQuery()
    consulta = "SELECT " + rorw + " FROM elementos WHERE ip_address='" + ip + "';"
    buscaComm.exec_(consulta)
    size = 0
    while buscaComm.next():
        size += 1
        if size > 1:
            QMessageBox.warning(None, "Error", "Tiene usted un problema de IP
duplicada. Por favor, solucionelo.")
            return
        else:
            try:
                if buscaComm.value(0).isNull:
                    tree = ET.parse('config.xml')
                    rorw = rorw.strip('c')
                    rorw = 'C' + rorw
                    for item in tree.iter():
                        if item.tag == rorw:
                            comm = item.text
            except:
                comm = buscaComm.value(0)
    return comm
```

Apéndice D

ventanaPrincipal.pyw

```
#!/usr/bin/env python3.5
# -*- coding: utf-8 -*-
# Fichero: ventanaPrincipal.pyw
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
#############################################################################
# Descripción: El objetivo del siguiente fichero es representar la          #
# ventana principal del Trabajo Fin de Grado "Herramienta gráfica de        #
# gestión SNMP y manejo de MIB". En ésta, incluiremos un Toolbar con        #
# cinco distintas opciones y un widget central con dos ventanas que         #
# albergarán dos listas distintas: una para mostrar los elementos           #
# gestionados y la otra para mostrar las alarmas.                           #
#############################################################################

# Importaciones generales
import platform
import sys
import threading
import time

# Importaciones de PyQT
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4.QtSql import *

# Importaciones de nuestra aplicación
import qrc_recursos # Proporcionamos una ruta inequívoca para nuestros recursos
import controlador # Llamadas al controlador central
import editarConf # Para llamar al diálogo destinado a modificar la configuración
import loadMib # Para realizar las cargas de MIB
import busquedas # Para llamar al diálogo destinado a efectuar búsquedas
import dialogosAdicionales # Diálogos como por ejemplo el que se ha de abrir tras pulsar
      sobre un elemento

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
#############################################################################
#                        Historial de versiones                             #
# ------------------------------------------------------------------------- #
# Versión 1.0.0: 5 de marzo de 2018                                         #
#                    Interfaz gráfica inicial                               #
# Versión 1.0.1: 9 de marzo de 2018                                         #
#                    Adición del formato tabla para el centralWidget.       #
#                    Conexión con la opción de modificarConf.               #
# Versión 1.0.2: 12 de marzo de 2018                                        #
#                    Conexión con la opción de carga de MIBs.               #
#                    Carga inicial de MIBs.                                 #
# Versión 1.0.3: 15 de marzo de 2018                                        #
#                    Lecturas a la base de datos (inicial y tras            #
#                    evento) a fin de presentar la tabla de elementos       #
#                    gestionados al usuario.                                #
# Versión 1.0.4: 19 de marzo de 2018                                        #
#                    Conexión con la opción de búsqueda.                    #
```

```
# Versión 1.0.5: 22 de marzo de 2018                                    #
#                    Conexión con el diálogo SetNextWalk.               #
#                    Conexión con el diálogo ArbolDlg.                  #
#                    Traslado de la respuesta del getNext a ArbolDlg.   #
#                    Conexiones del walk completo con el árbol.         #
# Versión 1.0.6: 25 de marzo de 2018                                    #
#                    Conexiones de snmpset funcionales.                 #
# Versión 1.0.7: 30 de marzo de 2018                                    #
#                    Receptor de traps desarrollado.                    #
# Versión 1.0.8: 1 de abril de 2018                                     #
#                    Modificaciones debidas al cambio de BBDD a SQLite. #
#                    Supresión de la opción "Editar valores" de la barra #
#                    de herramientas.                                   #
#                    Adición de la opción "About" a la barra de         #
#                    herramientas.                                      #
# Versión 1.0.9: 2 de abril de 2018                                     #
#                    Receptor de traps sin fichero auxiliar.            #
# Versión 1.1.0: 10 de abril de 2018                                    #
#                    Primera versión totalmente operativa.              #
#########################################################################

#########################################################################
#                            Clase: MainWindow                          #
#                                                                       #
# Se trata de la ventana principal de la aplicación. Está compuesta, en #
# orden vertical, por una barra de herramientas, una tabla de elementos #
# gestinados (que conforma el centralWidget), otra tabla para las       #
# alarmas (realizada con un dockWidget) y una barra de estado. Sirve    #
# como nexo entre el resto de elementos de la vista y el controlador.   #
#########################################################################
class MainWindow(QMainWindow):

    # Inicialización de la ventana
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        # Instanciaciones iniciales
        self.miControlador = controlador.ControladorCentral()

        # Título de la ventana
        self.setWindowTitle("SNMP assistant Manager")

        # Tabla de elementos gestionados
        self.tablaElementos = QTableWidget()

        # Dibujamos la tabla del centralWidget para los elementos gestionados
        # Se deja por determinar el número de filas, en virtud de la lectura a la BBDD,
    así como los
        # elementos de la tabla y su contenido. Esto se realiza en el método
    updateTablaElementos()
        self.tablaElementos.setColumnCount(10)
        self.tablaElementos.setHorizontalHeaderLabels(["Dir. MAC", "Dir. IP", "Último
    contacto", \
                                                       "sysObjectID", "sysDescr", "
    sysUpTime", "sysContact", \
                                                       "sysName", "sysLocation", "
    sysServices"])
        self.tablaElementos.setAlternatingRowColors(True)
        self.tablaElementos.setEditTriggers(QTableWidget.NoEditTriggers)
        self.tablaElementos.setSelectionBehavior(QTableWidget.SelectRows)
        self.tablaElementos.setSelectionMode(QTableWidget.SingleSelection)
        self.updateTablaElememntos()

        # Tabla de alarmas
        self.tablaAlarmas = QTableWidget()
        self.tablaAlarmas.setColumnCount(3)
        self.tablaAlarmas.setHorizontalHeaderLabels(["Elemento afectado", "Trap", "
    Llegada"])
        self.tablaAlarmas.setAlternatingRowColors(True)
        self.tablaAlarmas.resizeColumnsToContents()

        # Lanzamos el método updateTablaAlarmas en segundo plano. Buscará actualizarse
    cada 10 segundos
        actualizarAlarmas = threading.Thread(target=self.updateTablaAlarmas, name="
    ActualizarTablaDeAlarmas", daemon=True)
        actualizarAlarmas.start()

        # CentralWidget
        self.setCentralWidget(self.tablaElementos)

        # DockWidget para las alarmas
        alarmasWidget = QDockWidget("Alarmas", self)
        alarmasWidget.setObjectName("AlarmasWidget")
```

```
        alarmasWidget.setAllowedAreas(Qt.BottomDockWidgetArea)
        alarmasWidget.setWidget(self.tablaAlarmas)
        self.addDockWidget(Qt.BottomDockWidgetArea, alarmasWidget)

        # Barra de herramientas
        toolbar = self.addToolBar("Toolbar_principal")
        toolbar.setObjectName("ToolbarPrincipal")

        # Buscar
        findAction = QAction(QIcon(":/find.png"), "&Find", self)
        findAction.setShortcut(QKeySequence.New)
        helptext = "Buscar"
        findAction.setToolTip(helptext)
        findAction.setStatusTip(helptext)
        self.connect(findAction, SIGNAL("triggered()"), self.abrirBusquedas)

        # Resincronizar
        resyncAction = QAction(QIcon(":/resync.png"), "&Resync", self)
        resyncAction.setShortcut(QKeySequence.New)
        helptext = "Resincronizar"
        resyncAction.setToolTip(helptext)
        resyncAction.setStatusTip(helptext)
        self.connect(resyncAction, SIGNAL("triggered()"), self.orderManualResync)

        # Modificar configuración
        editConfAction = QAction(QIcon(":/config.png"), "&Config", self)
        editConfAction.setShortcut(QKeySequence.New)
        helptext = "Editar configuración"
        editConfAction.setToolTip(helptext)
        editConfAction.setStatusTip(helptext)
        self.connect(editConfAction, SIGNAL("triggered()"), self.editConf)

        # Carga de MIBs
        mibUploadAction = QAction(QIcon(":/upload.png"), "&MIBUpload", self)
        mibUploadAction.setShortcut(QKeySequence.New)
        helptext = "Carga de MIBs"
        mibUploadAction.setToolTip(helptext)
        mibUploadAction.setStatusTip(helptext)
        self.connect(mibUploadAction, SIGNAL("triggered()"), self.mibUpload)

        # About
        aboutAction = QAction(QIcon(":/about.png"), "&About", self)
        aboutAction.setShortcut(QKeySequence.New)
        helptext = "Información de la aplicación"
        aboutAction.setToolTip(helptext)
        aboutAction.setStatusTip(helptext)
        self.connect(aboutAction, SIGNAL("triggered()"), self.helpAbout)

        # Añadimos los distintos iconos a la barra de herramientas
        toolbar.addAction(findAction)
        toolbar.addAction(resyncAction)
        toolbar.addAction(editConfAction)
        toolbar.addAction(mibUploadAction)
        toolbar.addAction(aboutAction)

        # Barra de estado
        self.sizeLabel = QLabel()
        self.sizeLabel.setFrameStyle(QFrame.StyledPanel | QFrame.Sunken)
        status = self.statusBar()
        status.setSizeGripEnabled(False)
        status.addPermanentWidget(self.sizeLabel)
        status.showMessage("Ready", 5000)

        # Recogemos la pulsación de un usuario sobre un elemento.
        # Recogemos la señal emitida por el método updateTablaAlarmas (ejecutado fuera
del
        # hilo principal) para actualizar la tabla de alarmas, llamando a
updateTablaAlarmas2,
        # que se ejecuta en el hilo principal y se encargará de dibujar la tabla de
alarmas actualizada.
        self.connect(self.tablaElementos, SIGNAL("cellDoubleClicked(int, int)"), self.
elementoSeleccionado)
        self.connect(self, SIGNAL("actualizaLaTablaDeAlarmas()"), self.
updateTablaAlarmas2)


#########################################################################
#                    Método: elementoSeleccionado                       #
#                                                                       #
# El siguiente método es notificado de la pulsación de un elemento      #
# a fin de lanzar el diálogo SetWalkNextDlg. Recogerá el resultado      #
# de la interacción del usuario con este diálogo y trasladará al        #
# controlador la petición del usuario, si la hubiere. Así mismo, se     #
```

```
# encargará de recibir la respuesta del controlador en caso          #
# procedente y desencadenar las operaciones de vista necesarias      #
# llegado el caso.                                                   #
######################################################################
def elementoSeleccionado(self):

    # Obtenemos la IP del elemento seleccionado y la enviamos como parámetro al
SetNextWalkDlg.
    macElemento = self.tablaElementos.item(self.tablaElementos.currentRow(), 0).text
()
    ipElemento = self.tablaElementos.item(self.tablaElementos.currentRow(), 1).text
()
    miDlgEleccion = dialogosAdicionales.SetWalkNextDlg(macElemento, ipElemento)
    miDlgEleccion.exec_()

    # Inspeccionamos el valor de los parámetros de control del diálogo, una vez ha
finalizado su ejecución.
    # Solicitamos al controlador la realización de la operación apropiada y
recogemos los resultados
    # para mostrarlos al usuario. En el caso de getNext y walk, el encargado para
este cometido será
    # DlgArbol. En el caso de las operaciones con la base de datos, actuamos
directamente.
    if miDlgEleccion.getNextOrdenado:
        valores, etiquetas = self.miControlador.peticiones("getnext", miDlgEleccion.
insercionModulo.text(), ipElemento)
        miArbol = dialogosAdicionales.DlgArbol(valores, etiquetas)
        miArbol.exec_()

    if miDlgEleccion.walkOrdenado:
        valores, etiquetas = self.miControlador.peticiones("walk", "", ipElemento)
        miArbol = dialogosAdicionales.DlgArbol(valores, etiquetas)
        miArbol.exec_()

    if miDlgEleccion.setOrdenado:
        self.miControlador.peticiones("set", miDlgEleccion.recogerOid.text(),
ipElemento, miDlgEleccion.recogerNuevoValor.text())

    if miDlgEleccion.deleteOrdenado:
        peti = "DELETE FROM elementos WHERE mac_address='" + macElemento + "';"
        peticion = QSqlQuery()
        peticion.exec_(peti)
        if peticion.numRowsAffected() != 0:
            QMessageBox.information(None, "¡Éxito!", "El elemento con dirección IP "
 + ipElemento + " y dirección"
                                                     "MAC " + macElemento + " ha
sido eliminado de la "
                                                                           "base
 de datos.")
            self.updateTablaElememntos()

    if miDlgEleccion.editConfOrdenada:
        peti = "UPDATE elementos SET communityRO='" + miDlgEleccion.recogerCommRO.
text() + "', communityRW='" + \
        miDlgEleccion.recogerCommRW.text() + "' WHERE mac_address='" + macElemento +
 "';"
        peticion = QSqlQuery()
        peticion.exec_(peti)
        if peticion.numRowsAffected() != 0:
            QMessageBox.information(None, "¡Éxito!", "Sus cambios de configuración
han sido correctamente"
                                                     " escritos en la base de datos.
")

    # miArbol.controlCambio nos informa sobre si algún elemento del árbol ha sido
modificado.
    # En ese caso, intentamos un snmpset sobre el o los OIDs cuyo valor se haya
modificado.
    # Obtenemos la ruta como lista de etiquetas, así que la parseamos para que sea
una cadena
    # con las etiquetas separadas por puntos, de forma que sea interpretable para
PySNMP. Así,
    # para sysName, enviaríamos al controlador lo siguiente:
    # 'iso.org.dod.internet.mgmt.mib-2.system.sysName.0'
    if miArbol.controlCambio:
        oid = ""
        i = 0
        for cambio in miArbol.cambios:
            for nodo in miArbol.cambiosRutas[i]:
                if oid == "":
                    oid = oid + nodo
                else:
                    oid = oid + "." + nodo
```

```
            self.miControlador.peticiones("set", oid, ipElemento, cambio[1])
            oid = ""
            i += 1

################################################################################
#                    Método: abrirBusquedas                                    #
#                                                                              #
# Crea una instancia del diálogo Byscar y la ejecuta, mostrando al             #
# usuario una ventana emergente en la que indicar la búsqueda que              #
# desea. En caso de que su petición de búsqueda sea exitosa, el                #
# resultado le será mostrado en forma de tabla.                                #
################################################################################
def abrirBusquedas(self):
    dialogo = busquedas.Buscar()
    dialogo.exec_()


################################################################################
#                    Método: orderManualResync                                 #
#                                                                              #
# Se encarga de realizar la conexión con el controlador para                   #
# ordenar una resincronización, una vez que el botón adecuado en               #
# la barra de herramientas es pulsado. Una vez terminada la                    #
# resincronización, ordenará la actualización de la lista de                   #
# elementos mostrada al usuario.                                               #
################################################################################
def orderManualResync(self):
    self.miControlador.resync()
    self.updateTablaElememntos()


################################################################################
#                    Método: editConf                                          #
#                                                                              #
# Crea una instancia del diálogo EditarConfDlg y la ejecuta,                   #
# mostrando al usuario una ventana emergente donde puede modificar             #
# la configuración de la aplicación.                                           #
################################################################################
def editConf(self):
    dialog = editarConf.EditarConfDlg()
    dialog.exec_()


################################################################################
#                    Método: mibUpload                                         #
#                                                                              #
# Crea una instancia del diálogo LoadMibDlgIni y la ejecuta,                   #
# mostrando al usuario una ventana emergente donde puede cargar                #
# distintos módulos o compiladores MIB.                                        #
################################################################################
def mibUpload(self):
    dialog = loadMib.LoadMibDlgIni(self.miControlador)
    dialog.exec_()


################################################################################
#                    Método: helpAbout                                         #
#                                                                              #
# Este método muestra el mensaje "About" en una ventana emergente.             #
# En ella se mostrará una breve descripción de la aplicación, su               #
# desarrollo y las tecnologías empleadas.                                      #
################################################################################
def helpAbout(self):
    QMessageBox.about(self, "Acerca de ValladoMIB", """<b>ValladoMIB</b> v %s <p>
Desarrollado por Gonzalo
    Lezcano Hermoso. Escuela Técnica Superior de Ingenieros de Telecomunicación.
Universidad de Valladolid.
    2018. <p>Esta aplicación corresponde al Trabajo Fin de Grado con título '
Herramienta gráfica de gestión
    SNMP y manejo de MIB'. <p>Python %s - Qt %s - PyQt %s en %s""" % (__version__,
platform.python_version(),
                                                                      QT_VERSION_STR,
PYQT_VERSION_STR, platform.system()))


################################################################################
#                    Método: updateTablaElementos                              #
#                                                                              #
# Se encarga de actualizar la tabla de elementos, desplegada sobre             #
# el centralWidget.                                                            #
# Lee el contenido existente en la tabla 'elementos', establece el             #
# número de filas del widget en función de la respuesta de la BBDD.            #
```

```
# Posteriormente, con la ayuda de un bucle, rellena la tabla con    #
# los valores devueltos a raíz de la consulta para dirección IP y   #
# MAC. Tras ello, trata de contactar vía controlador con los        #
# elementos de la base de datos, a fin de obtener unos datos para   #
# el módulo SNMPv2-MIB::system (que es mostrado) actualizados. Si   #
# su comunicación es exitosa, mostrará los resultados en el widget. #
# En caso contrario, se informa al usuario y se dejan estos campos  #
# en blanco. Además, se actualiza la hora de último contacto en la  #
# base de datos una vez dibujado, con los valores presentes en la   #
# tabla.                                                            #
#####################################################################
def updateTablaElememntos(self):

    # Realizamos la consulta
    query = QSqlQuery()
    query.exec_("SELECT * FROM elementos;")

    # Obtenemos el número de filas
    numFilas = 0
    while query.next():
        numFilas += 1
    self.tablaElementos.setRowCount(numFilas)

    # Como estábamos al final de la query tras el bucle que determina el número de
filas,
    # nos colocamos al principio para el bucle que rellena esas filas.
    query.seek(-1)
    i = 0
    while query.next():
        mac_address = QTableWidgetItem(query.value(0))
        ip_address = QTableWidgetItem(query.value(1))
        ip_addr = str(query.value(1))

        self.tablaElementos.setItem(i, 0, mac_address)
        self.tablaElementos.setItem(i, 1, ip_address)

        try:
            timeDetected, descr, objectID, upTime, contact, name, location, services
 = self.miControlador.wakeUp(ip_addr)
            lastTimeDetected = QTableWidgetItem(timeDetected)
            sysObjectID = QTableWidgetItem(objectID)
            sysDescr = QTableWidgetItem(descr)
            sysUpTime = QTableWidgetItem(upTime)
            sysContact = QTableWidgetItem(contact)
            sysName = QTableWidgetItem(name)
            sysLocation = QTableWidgetItem(location)
            sysServices = QTableWidgetItem(services)

            self.tablaElementos.setItem(i, 2, lastTimeDetected)
            self.tablaElementos.setItem(i, 3, sysObjectID)
            self.tablaElementos.setItem(i, 4, sysDescr)
            self.tablaElementos.setItem(i, 5, sysUpTime)
            self.tablaElementos.setItem(i, 6, sysContact)
            self.tablaElementos.setItem(i, 7, sysName)
            self.tablaElementos.setItem(i, 8, sysLocation)
            self.tablaElementos.setItem(i, 9, sysServices)

        except:
            QMessageBox.warning(None, "Advertencia", "El dispositivo con dirección
IP " + ip_addr + " no se"
                                                                    "encuentra
accesible desde este servidor.")
            lastTimeDetected = QTableWidgetItem(query.value(2))
            self.tablaElementos.setItem(i, 2, lastTimeDetected)

        i += 1

    self.tablaElementos.resizeColumnsToContents()

    # Además, como acabamos de detectar ciertos elementos, actualizamos el valor
lastTimeDetected
    # de la base de datos.
    j = 0
    while j<numFilas:
        mac = self.tablaElementos.item(j, 0).text()
        last = self.tablaElementos.item(j, 2).text()
        consulta = "UPDATE elementos SET lastTimeDetected='" + str(last) + "' WHERE
mac_address='" + str(mac) + "';"
        peticion = QSqlQuery()
        peticion.exec_(consulta)
        j += 1
```

```
    ##########################################################################
    #                      Método: updateTablaAlarmas                        #
    #                                                                        #
    # El presente método consiste en ejecutar un bucle infinito que,         #
    # cada 10 segundos, emite una señal para que se actualice la tabla       #
    # de alarmas. Se ejecuta en un hilo secundario para que el bucle         #
    # infinito no bloquee el flujo normal del programa. En cambio, se        #
    # ha de dibujar en el hilo principal. Es por ello que se emite la        #
    # señal, captada por un método del hilo principal.                       #
    ##########################################################################
    def updateTablaAlarmas(self):
        while (1):
            time.sleep(10)
            self.emit(SIGNAL("actualizaLaTablaDeAlarmas()"))


    ##########################################################################
    #                      Método: updateTablaAlarmas2                       #
    #                                                                        #
    # El presente método se encargará de dibujar la tabla de alarmas en      #
    # función del contenido de la variable listaTrapReceiver, de             #
    # SnmpServer. Será llamado periódicamente a fin de mantener esta         #
    # tabla actualizada.                                                     #
    # Inicialmente, y debido a las limitaciones de ser una managerlist,      #
    # se recorre entera para determinar su tamaño (no se dispone del         #
    # método len). De ahí se eliminan las 2 primeras líneas (preámbulo)      #
    # y se dibuja el resto del contenido de la lista de listas, que          #
    # alberga las traps recibidas durante la ejecución.                      #
    # Inicialmente se lee el fichero a fin de determinar el número de        #
    ##########################################################################
    def updateTablaAlarmas2(self):

        numLineas = 0
        for linea in self.miControlador.miServidor.listaTrapReceiver:
            numLineas += 1

        self.tablaAlarmas.setRowCount(numLineas-2)

        i = 0
        for linea in self.miControlador.miServidor.listaTrapReceiver:
            if linea[0] is 0:
                pass

            else:
                elementoAfectado = QTableWidgetItem(linea[0])
                valor = QTableWidgetItem(linea[1])
                llegada = QTableWidgetItem(linea[2])

                self.tablaAlarmas.setItem(i, 0, elementoAfectado)
                self.tablaAlarmas.setItem(i, 1, valor)
                self.tablaAlarmas.setItem(i, 2, llegada)

                i += 1

        self.tablaAlarmas.resizeColumnsToContents()


##############################################################################
# Función: Main                                                              #
##############################################################################
def main():
    app = QApplication(sys.argv)
    form = MainWindow()
    form.setMinimumSize(600, 600)
    form.show()
    app.exec_()


# Llamada a la función main()
main()
```

Apéndice E

editarConf.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Fichero: editarConf.py
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
#############################################################################
# Descripción: El objetivo del siguiente fichero es presentar un            #
# diálogo para alterar la configuración (contenida en el fichero            #
# config.xml) cuando el usuario así lo seleccione, clickando sobre el       #
# icono presente en la barra de herramientas, destinado a alterar la        #
# configuración.                                                            #
#############################################################################

# Importaciones generales
import sys

# Importaciones de PyQT
from PyQt4.QtCore import *
from PyQt4.QtGui import *

# Importaciones para lectura/escritura de config.xml
# cElemenTree es mucho más rápida y consume menos memoria que elemenTree
try:
    import xml.etree.cElementTree as ET
except importError:
    import xml.etree.ElementTree as ET

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
#############################################################################
#                          Historial de versiones                           #
# ------------------------------------------------------------------------- #
# Versión 1.0.0: 8 de marzo de 2018                                         #
#                  Versión inicial.                                         #
# Versión 1.0.1: 16 de marzo de 2018                                        #
#                  Adición de la carga de MIB como opción configurable.     #
# Versión 1.1.0: 10 de abril de 2018                                        #
#                 Primera versión totalmente operativa.                     #
#############################################################################

#############################################################################
#                          Clase: EditarConfDlg                             #
#                                                                           #
# La siguiente clase proporciona los métodos y atributos necesarios         #
# para dibujar el diálogo destinado a editar la configuración y su          #
# comunicación con el fichero de configuración (config.xml) a fin de        #
# escribir los cambios indicados por el usuario.                            #
#############################################################################
class EditarConfDlg(QDialog):

    # Inicialización. Abriremos el fichero de configuración, prepararemos
    # una lista para almacenar los módulos MIB a cargar (cuyo número es
    # desconocido), llamaremos a nuestro método particular para cargar el
```

```
# diálogo adecuado y estableceremos el título de la ventana emergente.
def __init__(self, parent=None):
    super(EditarConfDlg, self).__init__(parent)

    self.tree = ET.parse('config.xml')
    self.root = self.tree.getroot()
    self.mibUpload = []
    self.cargarDialogo()
    self.setWindowTitle("Modificar la configuración")


##############################################################################
#                          Método: cargarDialogo                             #
#                                                                            #
# Cargará el diálogo principal para editar la configuración. Será            #
# llamado al inicio de una instanciación de nuestra clase.                   #
##############################################################################
def cargarDialogo(self):

    # Recorremos el fichero config.xml y extraemos los parámetros relevantes.
    for elem in self.tree.iter():
        if elem.tag == 'Subnet':
            self.subnetIp = elem.text
        elif elem.tag == 'Prefix':
            self.subnetPrefix = elem.text
        elif elem.tag == 'CommunityRO':
            self.communityRO = elem.text
        elif elem.tag == 'CommunityRW':
            self.communityRW = elem.text
        elif elem.tag == 'MibUpload':
            for child in elem:
                self.mibUpload.append(child.text)

    # Etiquetas del diálogo
    etiqueta1 = QLabel("IP subred")
    etiqueta2 = QLabel("Máscara de subred")
    etiqueta3 = QLabel("Community Read-Only")
    etiqueta4 = QLabel("Community Read-Write")
    etiqueta5 = QLabel("Módulos MIB cargados al arranque")

    # Celdas del diálogo. Inicialmente presentarán los valores leídos
    # del fichero config.xml
    self.subnetIpField = QLineEdit(self.subnetIp)
    self.subnetPrefixField = QLineEdit(self.subnetPrefix)
    self.communityROField = QLineEdit(self.communityRO)
    self.communityRWField = QLineEdit(self.communityRW)

    # Botón "Añadir", para añadir más módulos MIB
    addButton = QPushButton("Añadir módulo")

    # Botones "Aplicar | Cerrar"
    buttonBox = QDialogButtonBox(QDialogButtonBox.Apply|QDialogButtonBox.Close)

    # Dibujamos el diálogo. Al no estar cerrado el número de módulos MIB
    # posibles, realizamos un bucle y dibujamos estos elementos y los posteriores
    # en función de un contador (i).
    grid = QGridLayout()
    grid.addWidget(etiqueta1, 0, 0)
    grid.addWidget(self.subnetIpField, 0, 1)
    grid.addWidget(etiqueta2, 1, 0)
    grid.addWidget(self.subnetPrefixField, 1, 1)
    grid.addWidget(etiqueta3, 2, 0)
    grid.addWidget(self.communityROField, 2, 1)
    grid.addWidget(etiqueta4, 3, 0)
    grid.addWidget(self.communityRWField, 3, 1)
    grid.addWidget(etiqueta5, 4, 0, 1, 2)
    i = 0
    self.listaCeldasMib = []
    for item in self.mibUpload:
        self.celdaTemporal = QLineEdit(self.mibUpload[i])
        self.listaCeldasMib.insert(i, self.celdaTemporal)
        grid.addWidget(self.celdaTemporal, i+5, 0, 1, 2)
        i += 1
    grid.addWidget(addButton, i+5, 1)
    grid.addWidget(buttonBox, i+6, 0, 1, 2)
    self.setLayout(grid)

    # Conectamos los botones a las señales apropiadas
    self.connect(buttonBox.button(QDialogButtonBox.Apply), SIGNAL("clicked()"), self
.apply)
    self.connect(buttonBox, SIGNAL("rejected()"), self, SLOT("reject()"))
    self.connect(addButton, SIGNAL("clicked()"), self.addField)
```

```
##########################################################################
#                          Método: apply                                 #
#                                                                        #
# Su cometido es el de recoger los valores introducidos por el           #
# usuario, comprobar los campos sensibles y escribir, en caso de         #
# que el resultado de las comprobaciones sea exitoso, los cambios        #
# correspondientes en el fichero config.xml                              #
##########################################################################
def apply(self):

    # Recogemos los valores de los distintos campos
    self.subnetIp = self.subnetIpField.text()
    self.subnetPrefix = self.subnetPrefixField.text()
    self.communityRO = self.communityROField.text()
    self.communityRW = self.communityRWField.text()

    # Ordenamos las comprobaciones de corrección apropiadas. En caso de que no
 satisfaga
    # los requerimientos será retornado un 1 por los métodos llamados.
    if self.comprobarIp() == 1:
        return
    if self.comprobarPrefix() == 1:
        return

    # Preparamos la escritura en nuestro fichero de configuración. Si alguno de los
 campos
    # destinados a los módulos MIB se encuentra vacío, se eliminará del fichero de
 configuración.
    for elem in self.tree.iter():
        if elem.tag == 'Subnet':
            elem.text = self.subnetIp
        elif elem.tag == 'Prefix':
            elem.text = str(self.subnetPrefix)
        elif elem.tag == 'CommunityRO':
            elem.text = self.communityRO
        elif elem.tag == 'CommunityRW':
            elem.text = self.communityRW
        elif elem.tag == 'MibUpload':
            i = 0
            for child in elem:
                if self.listaCeldasMib[i].text() is ' ':
                    elem.remove(child)
                else:
                    child.text = self.listaCeldasMib[i].text()
                i += 1

    # Escribimos los cambios apropiados en el fichero de configuración y cerramos la
  ventana.
    try:
        self.tree.write('config.xml')
        QMessageBox.information(self, "Configuración modificada", "Sus cambios han
sido correctamente "
                                                                  "escritos en el
fichero de configuración.")
        self.close()
    except Exception:
        QMessageBox.warning(self, "Error", "Su configuración no ha sido modificada
debido a algún error.")
        self.close()


##########################################################################
#                        Método: addField                                #
#                                                                        #
# Este método crea una instancia del diálogo destinado a añadir un       #
# nuevo módulo MIB al fichero de configuración y ejecutarlo. Una         #
# vez finalizada la ejecución de este diálogo, cerrará el principal.     #
# Si el usuario volviese a abrir el diálogo editConfDlg, podría ver      #
# el módulo recién insertado en el fichero config.xml                    #
##########################################################################
def addField(self):
    miAdicion = AddModuleDlg()
    miAdicion.exec_()
    self.close()


##########################################################################
#                        Método: comprobarIp                             #
#                                                                        #
# El objetivo de este método es determinar si el valor introducido       #
# por el usuario tiene el formato correcto. Se esperan direcciones       #
# IPv4. Se esperan por tanto 4 bytes en formato decimal separados        #
```

```
# por puntos ('.'), sin valores negativos y, por lo tanto, con un   #
# valor máximo de 255 (2^8 - 1) por octeto. Que un usuario          #
# introduzca un valor que, conforme al prefijo, no corresponda a    #
# una dirección de subred (por ejemplo, 192.168.100.130/24), no se  #
# comprobará en este método y se realizará la conversión a una      #
# dirección de subred correcta en el método resync de la clase      #
# ControladorCentral.                                               #
#####################################################################
def comprobarIp(self):

    # Inicializamos nuestras variables de control
    contadorBytes = 0
    contadorDigitos = 0
    octeto = ''

    # El campo no puede estar vacío
    if self.subnetIp is '' :
        QMessageBox.warning(self, "Error de formato", "El campo de dirección IP no
puede quedar vacío.")
        self.subnetIpField.setFocus()
        return 1

    # Si el campo no está vacío, lo recorremos caracter a caracter para determinar
su corrección
    for char in self.subnetIp:
        # Si el caracter no es un punto, deberíamos encontrarnos dentro de un byte
        if char is not ".":
            # Comprobamos que, de no tratarse de un '.', es un caracter numérico
            if char not in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']:
                QMessageBox.warning(self, "Error de formato", "La dirección IP y la
máscara deben"
                                                              " ser escritas en
formato decimal, con un"
                                                              " máximo de 3 dígitos
por byte")
                self.subnetIpField.selectAll()
                self.subnetIpField.setFocus()
                return 1
            contadorDigitos += 1

            # Comprobamos que no se han introducido más de 3 dígitos en el byte
            if contadorDigitos > 3:
                QMessageBox.warning(self, "Error de formato", "La dirección IP y la
máscara deben"
                                                              " ser escritas en
formato decimal, con un"
                                                              " máximo de 3 dígitos
por byte.")
                self.subnetIpField.selectAll()
                self.subnetIpField.setFocus()
                return 1
            octeto = octeto + char

            # Comprobamos que el valor del byte no es superior a 255
            if contadorDigitos == 3:
                if int(octeto) > 255:
                    QMessageBox.warning(self, "Error de formato", "Cada byte debe
tener un valor menor o igual"
                                                                  " a 255.")
                    self.subnetIpField.selectAll()
                    self.subnetIpField.setFocus()
                    return 1

        # Si el caracter es un punto nos encontramos ante una separación de bytes
        else:
            # Comprobamos si el byte anterior ha quedado vacío
            if contadorDigitos == 0:
                QMessageBox.warning(self, "Error de formato", "No puede haber ningún
 octeto"
                                                              " vacío. Ni en el
campo de dirección IP ni en el de máscara"
                                                              " de subred.")
                self.subnetIpField.selectAll()
                self.subnetIpField.setFocus()
                return 1

            else:
                # Si no ha quedado vacío, comprobamos que no es mayor de 255
                if int(octeto) > 255:
                    QMessageBox.warning(self, "Error de formato", "Cada byte debe
tener un valor menor o igual"
                                                                  " a 255.")
                    self.subnetIpField.selectAll()
```

```
                    self.subnetIpField.setFocus()
                    return 1

                # Si ha ido bien, incrementamos el contador de bytes, ponemos a 0 el contador de dígitos
                # y dejamos vacía la cadena octeto.
                contadorBytes += 1
                contadorDigitos = 0
                octeto = ""

            # Si el contador de bytes supera el 3 significará que se han introducido más de 4 bytes
            if contadorBytes > 3:
                QMessageBox.warning(self, "Error de formato", "La dirección IP y la máscara de subred"
                                                              " deben tener 4 bytes.")
                self.subnetIpField.selectAll()
                self.subnetIpField.setFocus()
                return 1


    ##############################################################################
    #                       Método: comprobarPrefix                              #
    #                                                                            #
    # El objetivo de este método es determinar si el valor introducido           #
    # por el usuario tiene el formato correcto. Se esperan prefijos              #
    # enteros entre 0 y 32. Que un usuario introduzca un valor que,              #
    # conforme a la dirección que se haya escrito, no corresponda a              #
    # una dirección de subred (por ejemplo, 192.168.100.130/24), no se           #
    # comprobará en este método y se realizará la conversión a una               #
    # dirección de subred correcta en el método resync de la clase               #
    # ControladorCentral.                                                        #
    ##############################################################################
    def comprobarPrefix(self):

        # Si el valor no es un entero recogeremos la excepción
        try:
            self.subnetPrefix = int(self.subnetPrefixField.text())
        except ValueError:
            QMessageBox.warning(self, "Error de formato", "El prefijo de subred debe ser un entero.")
            self.subnetPrefixField.selectAll()
            self.subnetPrefixField.setFocus()
            return 1

        # No es aceptable un valor por encima de 32
        if self.subnetPrefix > 32:
            QMessageBox.warning(self, "Error de formato", "La máscara puede tener un máximo de 32 bits a '1'.")
            self.subnetPrefixField.selectAll()
            self.subnetPrefixField.setFocus()
            return 1

        # No es aceptable un valor negativo
        elif self.subnetPrefix < 0:
            QMessageBox.warning(self, "Error de formato", "La máscara no puede tener un número negativo de bits a '1'.")
            self.subnetPrefixField.selectAll()
            self.subnetPrefixField.setFocus()
            return 1


##############################################################################
#                         Clase: AddModuleDlg                                #
#                                                                            #
# La siguiente clase proporciona los métodos y atributos necesarios          #
# para dibujar el diálogo destinado a insertar un nuevo módulo MIB en        #
# el fichero config.xml. Este diálogo será llamado únicamente desde el       #
# diálogo editConfDlg.                                                       #
##############################################################################
class AddModuleDlg(QDialog):

    def __init__(self, parent=None):
        super(AddModuleDlg, self).__init__(parent)

        # Elementos que conforman el diálogo
        etiqueta = QLabel("¿Qué módulo MIB quiere incluir?")
        self.insercionModule = QLineEdit("Escriba aquí el módulo a incluir...")
        buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        # Dibujo del Layout
        grid = QVBoxLayout()
```

```
        grid.addWidget(etiqueta, 0)
        grid.addWidget(self.insercionModule, 1)
        grid.addWidget(buttonBox, 2)
        self.setLayout(grid)
        self.setWindowTitle("Añadir nuevo campo MIB")

        # Dejamos el control sobre el campo de inserción
        self.insercionModule.selectAll()
        self.insercionModule.setFocus()

        # Conexiones
        self.connect(buttonBox, SIGNAL("accepted()"), self, SLOT("accept()"))
        self.connect(buttonBox, SIGNAL("rejected()"), self, SLOT("reject()"))


    ##############################################################################
    #                            Método: accept                                  #
    #                                                                            #
    # Este módulo tiene como objetivo introducir en el fichero de                #
    # configuración el módulo deseado por el usuario. Para ello,                 #
    # generará la etiqueta adecuada e insertará el subelemento a                 #
    # continuación de los módulos MIB existentes en el fichero de                #
    # configuración en ese momento.                                              #
    ##############################################################################
    def accept(self):
        try:
            tree = ET.parse('config.xml')
            for elem in tree.iter():
                if elem.tag == 'MibUpload':
                    i = 1
                    for child in elem:
                        i += 1
                    etiquetaXml = 'Mib' + str(i)
                    nuevaMib = ET.Element(etiquetaXml)
                    nuevaMib.text = self.insercionModule.text()
                    elem.append(nuevaMib)
                tree.write('config.xml')
            QMessageBox.information(self, "¡Conseguido!", "Tus módulos se han cargado
    satisfactoriamente.")
        except Exception:
            QMessageBox.warning(self, "Error", "Tu petición no ha podido ser atendida
    satisfactoriamente.")


# Hacemos que el diálogo se ejecute si es llamado desde la función main(), presente en
    ventanaPrincipal.pyw
if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = EditarConfDlg()
    form.show()
    app.exec_()
```

Apéndice F

snmpserver.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Fichero: snmpserver.py
# Autor: Gonzalo Lezcano Hermoso
# Fecha de última modificación: 10 de abril de 2018
#
################################################################################
# Descripción: El objetivo del siguiente fichero es albergar la clase   #
# SnmpServer y sus distintos métodos, que permitirán la interacción con #
# los agentes a través de los get, set, walk, getnext y trap-receiver.  #
################################################################################

# Importaciones generales
import datetime
from multiprocessing import Manager, Process

# Importaciones de PySNMP
from pysnmp.hlapi import *
from pysnmp import smi
from pysnmp.entity import engine, config
from pysnmp.carrier.asyncore.dgram import udp
from pysnmp.carrier import error
from pysnmp.entity.rfc3413 import ntfrcv

#Importaciones de PySMI
from pysmi.reader.localfile import FileReader
from pysmi.writer.pyfile import PyFileWriter
from pysmi.parser.smi import SmiV2Parser
from pysmi.codegen.pysnmp import PySnmpCodeGen
from pysmi.compiler import MibCompiler

# Importaciones de PyQt
from PyQt4.QtGui import *

# Importaciones para lectura/escritura de config.xml
# cElemenTree es mucho más rápida y consume menos memoria que elemenTree
try:
    import xml.etree.cElementTree as ET
except importError:
    import xml.etree.ElementTree as ET

__author__ = "Gonzalo Lezcano Hermoso"
__version__ = "1.1.0"
################################################################################
#                          Historial de versiones                              #
# ---------------------------------------------------------------------------- #
# Versión 1.0.0: 5 de marzo de 2018                                            #
#                  Métodos snmpget y snmpwalk.                                 #
# Versión 1.0.1: 9 de marzo de 2018                                            #
#                  Método snmpset.                                             #
# Versión 1.0.2: 14 de marzo de 2018                                           #
#                  Los métodos loadMib y loadCompiler pasan a ser              #
#                  parte de la clase SnmpServer                                #
# Versión 1.0.3: 22 de marzo de 2018                                           #
```

```
#                    Desarrollo del método snmpgetnext.                    #
#                   Método snmpwalk corregido y adaptado a necesidades.    #
# Versión 1.0.4: 30 de marzo de 2018                                        #
#                    Desarrollo del trap-receiver.                          #
#                    Unión de los métodos snmpgetnext y snmpwalk.           #
# Versión 1.0.5: 2 de abril de 2018                                         #
#                    Trap-receiver sin fichero auxiliar.                    #
# Versión 1.1.0: 10 de abril de 2018                                        #
#                    Primera versión totalmente operativa.                  #
#############################################################################

#############################################################################
#                          Clase: SnmpServer                                #
#                                                                           #
# Agrupará las funcionalidades necesarias para realizar la función de       #
# servidor SNMP en nuestra aplicación, a saber, get, walk, set y            #
# trap-receiver. En su inicialización, lee el fichero de configuración,     #
# cargando los módulos MIB indicados en éste.                               #
#############################################################################
class SnmpServer(object):

    # Inicialización
    def __init__(self):
        super(SnmpServer, self).__init__()

        # Abrimos un nuevo proceso, compartiendo con él una variable de tipo lista de
     listas
        # para almacenar las traps. Utilizamos para ello un objeto de la clase Manager,
     declarando
        # también la estructura de dicha listas, de cara a que puedan ser añadidas
    nuevas traps
        # a su llegada.
        manager = Manager()
        fila = manager.list(["0", "Dos", "Tres"])
        self.listaTrapReceiver = manager.list([fila, fila])
        trapReceiver = Process(target=self.trapReceiver, name="Trap-receiver", daemon=
    True, args=(self.listaTrapReceiver,))
        try:
            trapReceiver.start()
        except:
            QMessageBox.warning(None, "Error", "No se ha podido abrir el puerto de
    escucha de traps. "
                                               "Compruebe que dispone de los privilegios
     necesarios. "
                                               "El programa puede seguir funcionando,
    pero no recibirá "
                                               "traps.")

        self.mibCompiler = MibCompiler(SmiV2Parser(), PySnmpCodeGen(), PyFileWriter('/
    tmp/pysnmp/mibs'))

        # Para resoluciones
        self.mibBuilder = smi.builder.MibBuilder()
        self.mibViewController = smi.view.MibViewController(self.mibBuilder)

        # Leemos, de config.xml, los módulos MIB a cargar inicialmente, y llamamos a
    loadMib para que los cargue
        tree = ET.parse('config.xml')
        mibUpload = []
        for elem in tree.iter(tag='MibUpload'):
            mibUpload.append(elem.text)

        self.loadMib(*mibUpload)


    #########################################################################
    #                          Método: snmpget                              #
    #                                                                       #
    # Se encarga de realizar un get, tomando como argumentos el OID         #
    # objetivo (consulta), la dirección del agente objetivo (direccion)     #
    # y la community, que en este caso debería ser de sólo lectura.         #
    #########################################################################
    def snmpget(self, consulta, direccion, community):
        try:
            g = getCmd(SnmpEngine(), CommunityData(community), UdpTransportTarget((
     direccion, 161)),
                       ContextData(),ObjectType(ObjectIdentity(consulta)))
            errorIndication, errorStatus, errorIndex, varBinds = next(g)
        except:
            return

        if errorIndication:
            print(errorIndication)
```

```
    elif errorStatus:
        print('%s at %s' %(errorStatus.prettyPrint(), errorIndex and varBinds[int(
errorIndex) - 1][0] or '?'))
    else:
        for varBind in varBinds:
            return varBind


#############################################################################
#                          Método: snmpwalk                                 #
#                                                                           #
# Tomando como argumentos la dirección IP del agente y la community         #
# de lectura, realiza un walk completo o un walk de un módulo del           #
# elemento objetivo. Es invocado por el controlador a petición del          #
# usuario vía GUI y sus resultados son representados posteriormente         #
# en forma de árbol por la GUI. La elección sobre si se trata de un         #
# walk completo o uno relativo a un módulo en concreto se toma en           #
# función del parámetro lexic. Si éste es 'True', el walk es                #
# completo, mientras que si es 'False' tomará el módulo indicado en         #
# "argumento" para realizar el walk sobre él. En caso de ser                #
# invocado un walk completo, este parámetro no será pasado y, por           #
# tanto, lo introduciremos nosotros como 'SNMPv2-MIB', lo que, en           #
# conjunto al lexicographicMode a 'True', y debido a un                     #
# comportamiento inesperado de PySNMP, nos devolverá el árbol               #
# completo.                                                                 #
#############################################################################
def snmpwalk(self, direccion, community, lexic, *argumento):

    # Almacenaremos en 'labels' una lista de listas con la identificación de cada
OID.
    # Por ejemplo, para sysDescr, obtendríamos:
    # ('iso', 'org', 'dod', 'internet', 'mgmt', 'mib-2', 'system', 'sysDescr').
    # Esto es logrado gracias a getLabel() y será usado para la representación en
forma de árbol.
    # En la lista valores almacenaremos los valores de estos elementos en un formato
 legible
    # para el ser humano, gracias a prettyPrint().
    labels = []
    valores = []

    if not argumento:
        argumento = ('SNMPv2-MIB',' ')

    for (errorIndication,
         errorStatus,
         errorIndex,
         varBinds) in nextCmd(SnmpEngine(), CommunityData(community),
UdpTransportTarget((direccion, 161)),
                              ContextData(), ObjectType(ObjectIdentity(argumento[0])
), lexicographicMode=lexic):
        if errorIndication:
            print(errorIndication)
            break
        elif errorStatus:
            print('%s at %s' %(errorStatus.prettyPrint(), errorIndex and varBinds[
int(errorIndex) - 1][0] or '?'))
            QMessageBox.warning(None, "Error", "Ha ocurrido algún problema al
realizar el walk. Revise"
                                               " su operación o el estado del agente
.")
            break
        else:
            for name, val in varBinds:
                name.resolveWithMib(self.mibViewController)
                labels.append(name.getLabel())
                valores.append(val.prettyPrint())
    return valores, labels


#############################################################################
#                          Método: snmpset                                  #
#                                                                           #
# El cometido de este método es alterar el valor de un OID de un            #
# agente determinado. Toma como parámetros de entrada el OID objeto         #
# de la operación (argumento), la dirección IP del elemento                 #
# gestionado (direccion), la community de lectura/escritura                 #
# (community) y el nuevo valor que debe ser escrito (nuevoValor).           #
#############################################################################
def snmpset(self, argumento, direccion, community, nuevoValor):
    try:
        errorIndication, errorStatus, errorIndex, varBinds = next(
            setCmd(SnmpEngine(),
                   CommunityData(community),
```

```
                    UdpTransportTarget((direccion, 161)),
                    ContextData(),
                    ObjectType(ObjectIdentity(argumento), nuevoValor)))
    except error.SmiError:
        QMessageBox.warning(None, "Error", "Revise el OID introducido, no parece ser
 correcto.")
        return

    if errorIndication:
        QMessageBox.warning(None, "Error", str(errorIndication))
    elif errorStatus:
        QMessageBox.warning(None, "Advertencia", "El OID " +
                            str(errorIndex and varBinds[int(errorIndex) - 1][0] or '
?') +
                            " no admite escritura. Sus cambios no han tenido efecto"
)
    else:
        QMessageBox.information(None, "¡Éxito!", "Sus cambios han sido correctamente
 escritos en el agente")


#############################################################################
#                    Método: decodificarOID                                 #
#                                                                           #
# Este método se encarga de proporcionar acceso al método                   #
# resolveWithMib del objeto mibViewController, instanciado en la            #
# inicialización de SnmpServer.                                             #
#############################################################################
def decodificarOID(self, OID):
    return OID.resolveWithMib(self.mibViewController)


#############################################################################
#                        Método: loadMib                                    #
#                                                                           #
# Este método se encarga de cargar al objeto mibBuilder los módulos         #
# MIB pasados como entrada.                                                 #
#############################################################################
def loadMib(self, *args):
    try:
        for arg in args:
            self.mibCompiler.compile(str(arg))
        return True
    except smi.error.MibNotFoundError:
        QMessageBox.warning(None, "Error SMI", "Alguna de las MIB pasadas para
 cargar no ha podido ser cargada.")
        return False


#############################################################################
#                        Método: loadCompiler                               #
#                                                                           #
# Este método se encarga de añadir un compilador de módulos MIB             #
# adicional, tomando como ruta el argumento pasado como entrada.            #
#############################################################################
def loadCompiler(self, arg):
    try:
        self.mibCompiler.addSources(FileReader(arg))
    except smi.error.MibNotFoundError:
        QMessageBox.warning(None, "Error SMI", "Alguna de las MIB pasadas para
 cargar, no ha podido ser cargada.")


#############################################################################
#                        Método: trapReceiver                               #
#                                                                           #
# Se ejecuta en segundo plano. Crea un servicio que escucha en el           #
# puerto UDP 162 a la espera de la llegada de traps. Con la llegada         #
# de una de estas trampas, añade esta, con el formato adecuado, a           #
# una lista de listas compartida por el proceso principal. Esta             #
# lista será posteriormente leída por la ventana principal, a fin           #
# de representar las llegadas de una forma "user-friendly".                 #
#############################################################################
def trapReceiver(self, listaRecibida):

    # Creamos un 'SNMP engine' con un engineID autogenerado y pre-asociado a un
 socket.
    snmpEngine = engine.SnmpEngine()

    # UDP sobre IPv4, escuchando en *:162
    try:
        config.addTransport(snmpEngine, udp.domainName + (1,), udp.UdpTransport().
openServerMode(('', 162)))
```

```
    except error.CarrierError:
        print("No ha podido ser abierto el puerto 162 para la recepción de traps."
              " El programa seguirá funcionando, pero no podrá recibir traps.")
        return

    # SecurityName <-> CommunityName mapping
    config.addV1System(snmpEngine, 'my-area', 'public')

    # Función Callback para recibir las notificaciones (traps)
    # noinspection PyUnusedLocal,PyUnusedLocal
    def cbFun(snmpEngine, stateReference, contextEngineId, contextName, varBinds,
cbCtx):

        # Contexto de ejecución
        execContext = snmpEngine.observer.getExecutionContext('rfc3412.
receiveMessage:request')

        # Obtenemos la dirección IP del agente que envía la notificación
        agente = str(execContext['transportAddress'][0])

        # Recorremos la notificación y la guardamos. Además, generamos una marca
temporal
        # y la guardamos también con el resto de los campos de la trampa
        # (agente, trap, hora). Así se representará en la ventana principal
        for name, val in varBinds:
            trampa = []
            trapPretty = str(name.prettyPrint()) + " = " + str(val.prettyPrint())
            trampa = [agente, trapPretty]
            trampa.append(str(datetime.datetime.today()))
            listaRecibida.append(trampa)

    # Lo registramos en el SNMP Engine
    ntfrcv.NotificationReceiver(snmpEngine, cbFun)

    # Iniciamos un trabajo que no acabará nunca (pues escuchamos siempre)
    # Esta es la razón por la que corre en otro hilo de ejecución.
    snmpEngine.transportDispatcher.jobStarted(1)

    # Corremos el I/O dispatcher que recibirá las traps y enviará respuestas (if any
)
    try:
        snmpEngine.transportDispatcher.runDispatcher()
    except:
        snmpEngine.transportDispatcher.closeDispatcher()
        raise
```

Bibliografía

[1] Ilya Etingof, PySNMP, «SNMP history», 2017 [Online]. Disponible en: `http://pysnmp.sourceforge.net/docs/snmp-history.html`. Consultado por última vez el 26 de marzo de 2018.

[2] Erik C. Rose, *Vulnerabilities of Network Control Properties: An Example*, RFC 789, Enero de 1981.

[3] Douglas R. Mauro, Kevin J. Schmidt, *Essential SNMP*, 2ª edición, O'Reilly Media, Septiembre de 2005.

[4] William Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*, 3ª edición, Addison Wesley Longman, Inc., 1999.

[5] J. Galvin, K. McCloghrie, *Security Protocols for version 2 of the Simple Network Management Protocol (SNMPv2)*, Network Working Group, Trusted Information Systems, Hughes LAN Systems, Abril de 1993.

[6] K. McCloghrie, M. Rose, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, RFC 1213, Hughes LAN Systems, Inc., Performance Systems International, Marzo de 1991.

[7] Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, Dan Mackin, *UNIX and Linux Administration Handbook*, 5ª edición, Pearson Education, Inc., 2018

[8] Nokia Networks, *Nokia 7750 Service Router. Release 15.1*, Nokia Oyj, Febrero de 2018.

[9] Huawei Technologies Co. Ltd., *S7700 & S9700 Series Switches V200R008C00. Configuration Guide - Network Management and Monitoring*, 7ª edición, Huawei Technologies Co. Ltd., Noviembre de 2017.

[10] Cisco, *Cisco ASR 1000 Series Aggregation Services Routers. SNMP Configuration Guide*, Cisco, Enero de 2018.

[11] Nokia Networks, *NSP Network Services Platform. Network Functions Manager - Packet (NFM-P). Release 17.6. System Architecture Guide*, Nokia Corporation, Enero de 2017.

[12] Cisco, *Cisco Unified Communications Manager Managed Services Guide, Release 8.5*, Cisco, Enero de 2014.

[13] Net-SNMP, *tkmib - an interactive MIB browser for SNMP. V5.7.3. Man page*, Net-SNMP, Noviembre de 2006.

[14] Tobias Oetiker, «Tobi Oetiker's MRTG - The Multi Router Traffic Grapher», 2017. [Online]. Disponible en `https://oss.oetiker.ch/mrtg/`. Consultado por última vez el 5 de mayo de 2018

[15] V. Cerf, R. Kahn, *A Protocol for Packet Network Intercommunication*, IEEE Transactions on Communications, COM-22(5), Mayo de 1974.

[16] Kevin R. Fall, W. Richard Stevens, *TCP/IP Illustrated. Volume 1: The Protocols*, 2ª edición, Pearson Education, Inc., 2012.

[17] M. Willet, R. D. Martin, *LAN management in an IBM framework*, IEEE Network: The Magazine of Global Internetworking, Volume 2 Issue 2, IBM Corp. Research Triangle Park, NC, Marzo de 1988.

[18] William Stallings, *SNMP, SNMPv2 and CMIP: the practical guide to network management standards*, Addison-Wesley Publishing Company, 1993.

[19] Charles M. Kozierok, *The TCP/IP Guide*, No Starch Press, Inc., 2005.

[20] J. Davin, J. Case, M. Fedor, M. Schoffstall, *A Simple Gateway Monitoring Protocol*, RFC 1028, Universidad de Tenessee en Knoxville, Universidad Cornell e Instituto Politécnico de Rensselaer, Noviembre de 1987.

[21] J. Case, M. Fedor, M. Schoffstall, J. Davin, *A Simple Network Management Protocol*, RFC 1067, Universidad de Tenessee en Knoxville, NYSERnet Inc., Instituto Politécnico de Rensselear, Proteon, Inc., Agosto de 1988.

[22] M. Rose, K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based internets*, RFC 1065, TWG, Agosto de 1988.

[23] K. McCloghrie, M. Rose, *Management Information Base for Network Management of TCP/IP-based internets*, TWG, Agosto de 1988.

[24] J. Case, M. Fedor, M. Schoffstall, C. Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1098, Network Working Group, Universidad de Tenessee en Knoxville, NYSERNet, Inc., Instituto Politécnico de Rensselaer, MIT Laboratory for Computer Science, Abril de 1989.

[25] J. Case, M. Fedor, M. Schoffstall, C. Davin, *A Simple Network Management Protocol (SNMP)*, RFC 1157, Network Working Group, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, Mayo de 1990.

[26] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, *Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)*, Network Working Group, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, Abril de 1993.

[27] K. McCloghrie, D. Perkins, J. Schoenwaelder, J. Case, M. Rose, *Structure of Management Information Version 2 (SMIv2)*, RFC 2578, Network Working Group, Cisco Systems, SNMPinfo, TU Braunschweig, SNMP Research, First Virtual Holdings, International Network Services, Abril de 1999.

[28] Ubizen AETHIS, *Security in SNMPv3 versus SNMPv1 or v2c*, SNMP Research International Inc., 2002.

[29] D. Harrington, R. Presuhn, B. Wijnen, *An Architecture for Describing SNMP Management Frameworks*, Network Working Group, Cabletron Systems, Inc., BMC Software, Inc., IBM T. J. Watson Research, Enero de 1998.

[30] M. St. Johns, *Diffie-Helman USM Key. Management Information Base and Textual Convention*, RFC 2786, Network Working Group, Marzo del 2000.

[31] U. Blumenthal, B. Wijnen, *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, RFC 3414, Network Working Group, Lucent Technologies, Diciembre de 2002.

[32] B. Wijnen, R. Presuhn, K. McCloghrie, *View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*,

RFC 3415, Network Working Group, Lucent Technologies, BMC Software, Inc., Cisco Systems, Inc., Diciembre de 2002

[33] Viptela, Cisco Systems, «SD-WAN Viptela. How it works», 2018 [Online]. Disponible en `http://viptela.com/sd-wan/`. Consultado por última vez el 6 de abril de 2018.

[34] U. Blumenthal, F. Maino, K. McCloghrie, *The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model*, RFC 3826, Network Working Group, Lucent Technologies, Andiamo Systems, Inc., Cisco Systems, Inc., Enero de 2004.

[35] D. Harrington, J. Salowey, W. Hardaker, *Secure Shell Transport Model for the Simple Network Management Protocol (SNMP)*, RFC 5592, Network Working Group, Huawei Technologies Estados Unidos, Cisco Systems, Cobham Analytic Solutions, Enero de 2009.

[36] , K. Narayan, D. Nelson, *Remote Authentication Dial-In User Service (RADIUS) Usage for Simple Network Management Protocol (SNMP) Transport Models*, RFC 5608, Network Working Group, Cisco Systems, Inc., Elbrys Networks, Inc., Agosto de 2009.

[37] W. Hardaker, *Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)*, RFC 6353, Internet Engineering Task Force (IETF), SPARTA, Inc., Julio de 2011.

[38] J. Schoenwaelder, *Simple Network Management Protocol (SNMP) over Transmission Control Protocol (TCP) Transport Mapping*, RFC 3430, Network Working Group, TU Braunschweig, Diciembre de 2002.

[39] A. Ben, A. Chadna, U. Warrier, «Network Management of TCP = IP Networks. Present and Future», *IEEE Network Magazine*, Julio de 1990.

[40] G. Waters, *User-based Security Model for SNMPv2*, RFC 1910, Network Working Group, Bell-Northern Research Ltd., Febrero de 1996.

[41] Paul Van de Zande, *The Day DES Died*, SANS Institute Reading Room, Julio de 2001.

[42] Net-SNMP Project, «Net-SNMP. About», 2013. [Online]. Disponible en `https://www.net-snmp.org/` Consultado por última vez: 9 de abril de 2018.

[43] Net-SNMP Project, «Net-SNMP. History», 2011. [Online]. Disponible en `https://www.net-snmp.org/about/history` Consultado por última vez: 9 de abril de 2018

[44] Hewlett-Packard Enterprise, *SNMP Manager Programmer's Guide*, Hewlett-Packard Enterprise, Mayo de 1997.

[45] Pierre Carbonelle, «PYPL PopularitY of Programming Language», GitHub, [Online]. Disponible en `https://pypl.github.io/PYPL.html` Consultado por última vez: 11 de abril de 2018

[46] netsnmpj, SourceForge, «Netsnmpj. About», SourceForge, [Online], Disponible en `https://netsnmpj.sourceforge.net` Consultado por última vez: 11 de abril de 2018

[47] Franck Fock, Jochen Katz, «SNMP4J - The Object Oriented SNMP API for Java Managers and Agents». SNMP4J.org, 2015, [Online], Disponible en: `https://www.snmp4j.org/index.html` Consultado por última vez: 11 de abril de 2018

[48] J. Jeffrey Hanson, *Pro JMX: Java Management Extensions*, APress Media, LLC, 2004.

[49] Ilya Etingof, PySNMP, «SNMP library for Python», PySNMP, 2017, [Online], Disponible en `https://pysnmp.sourceforge.net/`, Consultado por última vez: 11 de abril de 2018

[50] PyQt, SourceForge, «QTreeWidget Class Reference», PyQt, 2015, [Online] Disponible en: `http://pyqt.sourceforge.net/Docs/PyQt4/qtreewidget.html`, Consultado por útlima vez: 11 de abril de 2018.

[51] PyQt, SourceForge, «QTableWidget Class Reference», PyQt, 2015, [Online], Disponible en: `http://pyqt.sourceforge.net/Docs/PyQt4/qtablewidget.html`, Consultado por última vez: 11 de abril de 2018.

[52] Mark Summerfield, *Rapid GUI Programming with Python and Qt*, Prentice Hall, Pearson Education, Inc., 2008.

[53] SQLite, «Distinctive Features Of SQLite», SQLite, [Online], Disponible en: `https://www.sqlite.org/different.html`, Consultado por última vez: 11 de abril de 2018.

[54] SQLite, «SQLite Is Serverless», SQLite, [Online], Disponible en: `https://www.sqlite.org/serverless.html`, Consultado por última vez: 11 de abril de 2018.

[55] R. S. Pressman, *Ingeniería de Software. Un enfoque práctico*, 5ª edición, McGraw Hill, 2002.

[56] Shari Lawrence Pfleeger, Joanne M. Atlee, *Software Engineering*, Pearson Education, Inc., 4ª edición, 2010.

[57] Bernd Bruegge, Allen H. Dutoit, *Object-Oriented Software Engineering. Using UML, Patterns, and Java*, Pearson Education, Inc., Prentice Hall, 3ª edición, 2010.

[58] Ian Sommerville, *Software Engineering*, Addison-Wesley, Pearson Education Ltd., 6ª edición, 2001.

[59] Cisco y/o sus afiliados, *Cisco IOS LAN Switching Command Reference*, Cisco y/o sus afiliados, 2018.

[60] Santiago Millán Alonso, «Telefónica elige Segovia y Talavera para el futuro encendido del 5G en España», *El País. Cinco días*, 22 de enero de 2018.