INGENIERÍA DE DOMINIO ORIENTADA A OBJETOS APLICADA AL DESARROLLO DE UN FRAMEWORK PARA LA IMPLEMENTACIÓN DE AGENTES DE ADMINISTRACIÓN DE RED

TESIS

MAESTRÍA EN CIENCIAS CON ESPECIALIDAD EN TECNOLOGÍA INFORMÁTICA

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

POR:

CARLOS OMAR TORRES GONZÁLEZ

JUNIO DE 2003

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DIVISIÓN DE GRADUADOS EN COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

PROGRAMA DE POSGRADO EN COMPUTACIÓN, INFORMACIÓN Y COMUNICACIONES

Los miembros del comité de tesis recomendamos que la presente tesis del Ing. Carlos Omar Torres González sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias, especialidad en Tecnología Informática.

Comité de Tesis:

Dr. José Raúl Pérez Cázares, Ph. D.
ASESOR PRINCIPAL

Ing. Jaime Martínez Garza
SINODAL

Ing. Mario Isidro de la Fuente Martínez
SINODAL

Dr. David Alejandro Garza Salazar, Ph. D.
Director de los Programas de Posgrado en
Computación, Información y Comunicaciones

Junio de 2003

INGENIERÍA DE DOMINIO ORIENTADA A OBJETOS APLICADA AL DESARROLLO DE UN FRAMEWORK PARA LA IMPLEMENTACIÓN DE AGENTES DE ADMINISTRACIÓN DE RED

POR

CARLOS OMAR TORRES GONZÁLEZ

TESIS

Presentada a la División de Graduados en Computación, Información y Comunicaciones.
Este Trabajo es Requisito Parcial para Obtener el Grado de Maestro en Ciencias con Especialidad en Tecnología Informática.

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS MONTERREY

JUNIO DE 2003

DEDICATORIA

A Dios, por ser el ser supremo que guía mi existencia, a quien le dedico mis éxitos y fracasos.

A mi Papá, Eusebio Torres Trujillo, quien gracias a su amor, cuidado, gran sacrificio y visión me ha permitido ser un profesionista, cosa que le agradeceré siempre.

A mi Mamá, María Antonia González Galavíz, por traerme al mundo, por su sacrificio y gran amor que nos tiene a mi hermano y a mí, y por inculcarnos siempre la superación personal constante.

A mi querido Abuelo, Gerardo González Torres†, al cual siempre recordaré con mucho cariño ya que lo considero como mi segundo padre. Su última bendición me acompaña día a día.

A mi querida Abuelita, María Galavíz Martínez, a quien quiero mucho y cuyas bendiciones me ayudan en los momentos difíciles.

A mi hermano, Gerardo Eusebio Torres González, a quien estimo y quiero bastante.

A mis Tíos, Rosa María González, José Concepción Coronado, Roberto, Ma. del Carmen, Julia, Yolanda y Juan Galavíz Cruz, a mis primos José, Jorge y Mario, por su cariño y estimación, quienes hacen que nuestra familia siga unida, lo cual es la herencia más grande que nos dejó mi Abuelo.

A Silvia Tello Zúñiga por su amor, el apoyo moral y sentimental que ha tenido conmigo, y que ha hecho que superemos juntos muchas dificultades.

AGRADECIMIENTOS

Expreso mis más sincero agradecimiento a las todas las personas que colaboraron de alguna manera en el logro de la presente tesis.

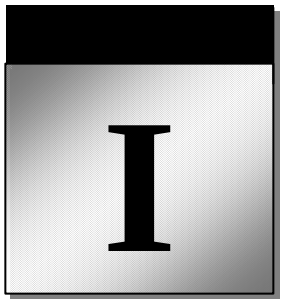
Al Dr. José Raúl Pérez Cázares, por su asesoría, apoyo, motivación, paciencia y disponibilidad para el desarrollo de la tesis.

Al Ing. Jaime Martínez Garza e Ing. Mario Izidoro de la Fuente Martínez, por su apoyo y tiempo dedicado a la revisión y evaluación del contenido de la tesis.

A mis maestros del ITESM, por compartir su sabiduría, conocimientos y guía, ya que sin ellos no hubiese obtenido la experiencia con la que hoy cuento.

Al Ing. Daniel Chávez, Ing. Luis Saldaña, Ing. Enrique Sibaja e Ing. Marisela Mora, por el gran apoyo brindado en los últimos meses del desarrollo de la tesis.

A todos mis amigos y maestros del Tecnológico de Ciudad Madero, por el apoyo moral que me brindaron en todo momento y sobre todo por su amistad.



Resumen

La industria de software está tendiendo más y más a promover el reuso de componentes de software previamente desarrollados. No obstante, la tecnología actual no cuenta todavía con herramientas suficientes para desarrollar rápida y eficientemente aplicaciones que se adapten a las necesidades requeridas. Es por ello que se ve la necesidad de generar componentes de software que impidan volver a inventar los conceptos y eventos presentes en un dominio de aplicaciones.

Una de las áreas en la que es necesario aplicar técnicas de reuso de software es en la Administración de Redes, ya que esta área en particular cuenta con software complejo y propenso a error, como lo son los Agentes y Administradores de Red, debido a que existen múltiples plataformas operativas donde este tipo de software es ejecutado, así también la gran diversidad de elementos de red, como lo son ruteadores, switches, multiplexores, etc., los cuales históricamente cuentan con software de administración de red propietario y no fácilmente extensible y reusable.

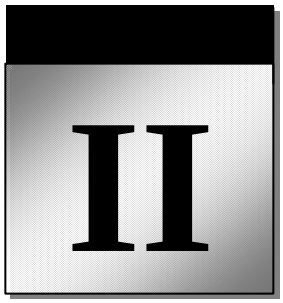
En la actualidad existen varias técnicas que permiten reutilizar software, lo cual impide que nuevamente se tengan que volver a inventar conceptos y eventos presentes en una nueva aplicación. Una de estas técnicas son *los Frameworks de Dominio Orientados a Objetos*, los cuales abstraen un dominio de aplicaciones en particular, y de esta forma, permiten desarrollar aplicaciones con una mayor rapidez y calidad, debido a que al utilizarlos ya no es necesario desarrollar aplicaciones desde cero, sino que son extendidos por las aplicaciones, lo cual disminuye considerablemente el tiempo de desarrollo.

No obstante que existe esta técnica de reuso, no es una tarea trivial el desarrollar un framework de dominio, por lo cual se necesita una metodología bien estructurada que permita implementarlos, es aquí donde es aplicada la *Ingeniería de Dominio Orientada a Objetos* ya que permite abstraer un dominio de en particular y obtener como producto final un framework que permita desarrollar una familia de aplicaciones.

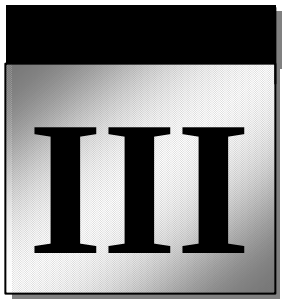
La presente tesis tiene como objetivo principal la aplicación de la Ingeniería de Dominio Orientada a Objetos en el dominio de los Agentes de Administración de Red, teniendo como producto final un framework de dominio, el cual permite desarrollar con mayor rapidez este tipo de aplicaciones complejas. El dominio de la tesis está delimitado por Agentes de Administración de Red que utilizan a los protocolos SNMP y CMIP para la comunicación de información hacia y desde el Administrador.

Las fases principales del desarrollo del framework contemplan el *Análisis de Dominio*, el cual permite obtener modelos conceptuales del dominio, el *Análisis de Variabilidad*, el cual es una propuesta por la presente tesis, que permite observar las similitudes y diferencias entre los conceptos del dominio, y el *Diseño de Dominio*, el cual mapea en clases, los conceptos obtenidos en el Análisis de Dominio, tomando en cuenta las similitudes y diferencias obtenidas en el Análisis de Variabilidad.

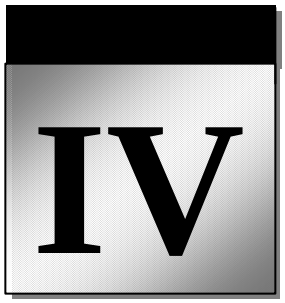
Cabe destacar que el desarrollo del framework de dominio no solo contempla el análisis y diseño del mismo, sino también su implementación así como el desarrollo de una aplicación Agente Proxy para el ruteador Cisco 2501.



Contenido



Indice de Tablas



Indice de Figuras

Capítulo

1

Introducción

1.1 Antecedentes

Las redes de telecomunicaciones día a día involucran dentro de sí a diversos elementos de red interactuantes, como lo son ruteadores, switches, hubs, etc., para poder soportar la gran demanda de servicios de red que existe en la actualidad. Esta demanda es debido a que más y más Usuarios requieren de éstos servicios para poder cubrir sus necesidades de información, como lo pueden ser teleconferencias, interacciones bancarias, grupos de discusión, correo electrónico, etc.

Un ejemplo claro de una amplia red de telecomunicaciones es Internet. Esta red mundial soporta solicitudes de información de millones de Usuarios que se conectan a ella diariamente. Actualmente Internet involucra a miles de redes, las cuales se conectan entre sí para brindar los servicios de información solicitados por los Usuarios, no por nada a Internet se le ha denominado la "red de redes". Ahora bien, cada una de éstas redes cuenta con el soporte de organizaciones o empresas que se encargan de planear, organizar y vigilar tanto que los servicios como los elementos que intervienen en su red atiendan de manera efectiva las solicitudes de información por parte de los Usuarios conectados a Internet. A éstas empresas se les denomina Proveedores de Servicios de Red (NSP de sus siglas en Inglés).

Para que un Usuario pueda utilizar los servicios de una red, antes debe existir una infraestructura, la cual es provista por un NSP. En este caso, es el NSP el encargado de proporcionar tanto el hardware como el software necesarios para permitir al Usuario conectarse a la red y hacer uso de sus múltiples servicios.

Los NSPs no sólo tienen que proporcionar la infraestructura, sino que además deben encargarse de monitorear constantemente todos los elementos de su red, esto con el fin de verificar que funcionen de manera adecuada, y en caso de detectar que algún elemento no realiza sus funciones correctamente, deben de solucionar el problema, de lo contrario podrían perder clientes. En este punto cabe hacer algunas preguntas, ¿Cómo es que los NSPs se dan cuenta que un elemento de su red no está funcionando adecuadamente?, ¿Cómo es que los NSPs determinan que se debe incrementar la capacidad de procesamiento

o almacenamiento de uno o varios elementos de red dada la demanda de Usuarios?. La respuesta no es sencilla, ya que involucra a una serie de entidades de software, servicios, protocolos etc., que permiten vigilar el estado de cada uno de los elementos de la red, además de realizar mediciones y estadísticas sobre el flujo de información que los elementos de la red reciben y envían. Todo esto se resume en un concepto ampliamente utilizado en las redes de telecomunicaciones, la "Administración de Redes".

La Administración de Redes propone una serie de protocolos, entidades de software y funciones de administración para monitorear, y controlar las actividades y los Recursos de una red. A través del intercambio de información de administración entre los dos roles principales en la Administración de Redes, el "Administrador" y el "Agente", es posible realizar el control y monitoreo sobre los Recursos. Tanto el Administrador como el Agente son entidades de software las cuales interactúan entre sí a través de un protocolo de administración de red. El Administrador es el encargado de realizar constantes solicitudes de actualización y obtención de información sobre Recursos, las cuales son atendidas por los Agentes que tiene a su cargo. El Agente a parte de despachar las solicitudes del Administrador, se encarga de reportarle eventos excepcionales ocurridos en un Recurso, los cuales se denominan alarmas. A veces las alarmas reflejan fallas, o cambios de estado en los Recursos los cuales pudieran recaer en un funcionamiento no adecuado en un elemento de red.

No obstante que tanto el Administrador como el Agente contienen la funcionalidad básica para poder verificar el estado de los Recursos de una Red, la Administración de Redes va más allá de ésta funcionalidad, ya que se divide principalmente en cinco niveles de abstracción, esta abstracción se refiere al tipo de información que se puede obtener al hacer interactuar a Administradores y Agentes. Según [Ranman, 1999], los niveles son los siguientes:

1. *Elemental*. El tipo de información utilizada en este nivel tiene que ver con los Recursos de un Elemento de Red, como lo puede ser el estado de los enlaces de un ruteador. En este caso se requiere la interacción de un Administrador y un Agente, con un solo intercambio de información de administración.
2. *Administración de Elemento de Red*. El tipo de información manejada en este nivel tiene que ver con el estado actual de un elemento de red, es decir, se ve al elemento de red como un todo. En este nivel se requiere la interacción de un Administrador y un Agente, no obstante, requiere varios intercambios de información, debido a que es necesario revisar todas las funcionalidades básicas del elemento de red.
3. *Administración de Red*. La información que se puede obtener en este nivel se refiere a los elementos que intervienen en un servicio punto a punto, es decir, si éstos están funcionando adecuadamente. Esto involucra a varios elementos de red, por lo cual requiere de la interacción de uno o varios Administradores y varios Agentes con un considerable intercambio de información.
4. *Administración de Servicio*. En este caso, se obtiene información a nivel servicio, solo interesa saber si un servicio puede mantener una calidad determinada (por ejemplo disponibilidad del 99.99%), sin preocuparse por los detalles técnicos o cómo es que se asegura ésta calidad de servicio.

5. *Administración de Negocio*. Este es el nivel más alto de abstracción. La información asociada se enfoca en los objetivos empresariales. En este caso se analiza que se hayan realizado correctamente las inversiones en la red, la cantidad de dinero redituado por cada servicio ofrecido por la empresa, las inversiones a realizar por la red, etc.

Como se puede observar, todos éstos niveles de abstracción de información son útiles para las diversas funciones desempeñadas por un NSP. A pesar de éstos niveles de abstracción, las funciones básicas siguen estando representadas por los Agentes y los Administradores, es decir, estas entidades de software son la base para la Administración de Redes, ya que son aquellos que directa o indirectamente permiten obtener la información necesaria en cada nivel.

Ahora bien, ¿Qué que tan fácil es el desarrollar software para proporcionar la funcionalidad de un Agente o un Administrador?. Según [Fayad, 1999], el diseño y la implementación de software que permiten administrar redes tiende a ser costoso y propenso a error. La gran cantidad del costo y esfuerzo recaen en el continuo redescubrimiento y reinvención de los conceptos y componentes principales que están involucrados en el desarrollo de software. En particular, la creciente heterogeneidad de arquitecturas de hardware y la diversidad de sistemas operativos y plataformas de comunicación hacen difícil el construir aplicaciones de administración de red correctas, portables, eficientes y económicas de la nada.

Si se desarrollan los Agentes o Administradores desde cero y sin una cultura de reuso de software, las posibles variaciones sobre elementos como el sistema operativo utilizado, el protocolo de administración de red, etc. necesariamente involucran volver a desarrollarlos [Aidarous, 1998].

Una solución para reuso de software en el desarrollo de Agentes o Administradores puede ser el generar librerías de funciones, las cuales contengan la funcionalidad requerida dependiendo del sistema operativo utilizado, el elemento de red, el protocolo de administración de red, etc. No obstante si alguno de éstos elementos varía, por ejemplo si es necesario utilizar otro protocolo de Administración de Red, y aún si se cuentan con las librerías necesarias para soportarlo, se tendría que empezar de nuevo con el desarrollo de las aplicaciones, ya que no se sabría en ese momento el gran impacto que esta variación tendría en todas la aplicaciones desarrolladas, y si éstas de alguna forma pudieran adaptarse para soportar el cambio. A parte, desarrollar y mantener este tipo de librerías es muy costoso en términos de tiempo de desarrollo.

Otra solución para el reuso de software es el desarrollar frameworks de aplicación orientados a objetos. [Fayad, 1999] comenta que los frameworks son una técnica de reuso orientada a objetos, los cuales promueven la cultura de desarrollo a mediano y largo plazo. Mediante la especialización de los mismos, permiten producir aplicaciones a la medida con un esfuerzo menor al de desarrollar la misma aplicación de la nada, es decir, permiten la creación de una familia de aplicaciones involucradas o inherentes en el dominio del framework.

Los frameworks orientados a objetos son una técnica de reuso ya que son éstos los que tienen el control y la lógica de negocio de las aplicaciones, el desarrollador de aplicaciones simplemente tiene que adaptar la funcionalidad requerida basándose en ésta lógica. Se adaptan las aplicaciones al framework basándose en los puntos de variabilidad definidos en el mismo, los cuales están representados por clases abstractas las cuales deben ser extendidas por la aplicación[1], ó por clases concretas del framework las cuales deben ser configuradas para su uso en el framework[2]. Los puntos de variabilidad no son más que funcionalidad presente en la lógica de negocio del framework, pero abstraída de tal forma que sea adaptable a la aplicación a desarrollar. Un ejemplo de un punto de variabilidad en el desarrollo de un Agente es el protocolo de Administración de Red a utilizar.

La clasificación de los frameworks acorde a las técnicas de extensión los divide en frameworks de caja blanca, frameworks de caja negra, y frameworks de caja gris:

- *Frameworks de caja blanca*. En este tipo de framework, la extensibilidad es conseguida mediante la utilización de características orientadas a objetos tales como la herencia y el enlace dinámico. La funcionalidad existente es reutilizada y extendida ya sea heredando de las clases abstractas del framework.
- *Frameworks de caja negra*. En un framework de caja negra la extensibilidad es lograda definiendo interfases para componentes que pueden ser "insertados" en el framework utilizando composición de objetos.
- *Frameworks de caja gris*. Un framework de caja gris es una mezcla entre el framework de caja blanca y el de caja negra. Los frameworks de caja gris permiten la extensibilidad utilizando herencia y enlace dinámico así como definiendo interfases.

Según [Fayad, 1999], los desarrolladores de aplicaciones en dominios complejos, tales como las telecomunicaciones, tradicionalmente han carecido de frameworks estándar. Como resultado, los desarrolladores en este dominio construyen, validan y mantienen sistemas de software de la nada, lo cual ha llegado a ser prohibitivamente costoso en términos de tiempo consumido en el desarrollo de aplicaciones. No obstante que existen numerosas herramientas para la administración de redes, éstas son la mayoría de las ocasiones no extensibles y construidas para un conjunto restringido de problemas y arquitecturas de red. Según [Fayad 2, 1999], se cree que la tecnología de frameworks de aplicación es una perfecta metodología para desarrollar y mantener sistemas estables de comunicación y administración de redes.

Dadas estos antecedentes, si se desarrollaran frameworks orientados a objetos para abstraer la funcionalidad de un Agente o un Administrador, permitiría que los NSPs puedan tener una base de desarrollo muy amplia lo cual conduciría a tener que invertir menos tiempo y capital en la creación de aplicaciones de administración de red.

---

[1] A éste tipo de reuso se le denomina de caja blanca, ya que el desarrollador debe conocer la interfaz proporcionada por el framework además de la funcionalidad esperada en cada método a definir.

[2] A éste tipo de reuso se le denomina de caja negra, ya que existen clases concretas en el framework, las cuales solo deben ser configuradas para la que la lógica de negocio del framework las utilice.

1.2 Objetivo de la Tesis

Dada la importancia del papel que tienen los Agentes y los Administradores, la cual se encuentra reflejada en la sección de Antecedentes, y dada la problemática que existe para desarrollar estas entidades de software debido a la gran diversidad de elementos y funcionalidad que deben manejar (Elementos de red, protocolos de administración de red, etc.), es necesario aplicar una técnica de reuso de software para de ésta forma agilizar el desarrollo de éste tipo de aplicaciones, por lo cual, el objetivo de la presente tesis es el analizar y diseñar un framework orientado a objetos que permita desarrollar familias de aplicaciones que tomen el papel de Agente en el contexto de la Administración de Red. El framework permitirá una gran flexibilidad en el desarrollo de Agentes, además de todas las ventajas que implica la utilización del framework, las cuales son:

- *Modularidad*. Los frameworks permiten encapsular detalles de implementación y lógica de negocio en clases abstractas, las cuales deben ser extendidas o adaptadas de tal forma que los módulos que se deben adaptar en el framework están identificados.
- *Reusabilidad*. Las interfases provistas por los frameworks permiten el reuso debido a que están definidos componentes genéricos que pueden ser nuevamente aplicados para crear nuevas aplicaciones.
- *Extensibilidad*. Un framework permite extensibilidad proveyendo los puntos de variabilidad reflejados en métodos abstractos, los cuales permiten extender el framework para formar la aplicación deseada.
- *Inversión de Control*. A través de la inversión de control, la lógica de negocio está ya definida por el framework, esto es, los pasos necesarios para ejecutar los servicios de una aplicación ya están determinados y controlados por el framework, lo cual permite simplemente enfocarse en adaptarlo para formar la nueva aplicación, sin necesidad de implementar la lógica de negocio.

Los Agentes desarrollados podrán interpretar y dar seguimiento a servicios solicitados por un Administrador que utilice ya sea el Protocolo Común de Información de Administración (CMIP de sus siglas en Inglés) o el Protocolo Simple de Administración de Red (SNMP de sus siglas en Inglés). Se toma en cuenta SNMP porque es el protocolo de Administración de Red más utilizado en la actualidad, y CMIP, que aunque su uso es limitado, está orientado a implementarse en grandes elementos de red que tengan una capacidad de procesamiento considerable. Ambos protocolos son descritos en el siguiente capítulo (Véase el "*Capítulo 2. Agentes de Administración de Red*" en su "*Sección 2.4. Protocolos de Administración de Red*").

Basándose en la extensibilidad del framework, se modelará los recursos de un elemento de red a través de Objetos Administrados, los cuales sean independientes del modelo de información a utilizar (OSI o Internet), lo cual forma permitirá formar una amplia gama de Bases de Información de Administración (MIB de sus siglas en inglés.) las cuales se componen de Objetos Administrados. Tanto los Objetos Administrados como las MIBs son descritos también en el siguiente capítulo (Véase el "*Capítulo 2*" en su "*Sección 2.5. Base de Información de Administración*").

El framework además permitirá abstraer una entidad lógica que tenga interacción directa con los recursos de un elemento de red, denominada Administración Virtual de Hardware (AVH), la cual se encargará de monitorear el estado de los recursos de un elemento de red de acuerdo a los eventos presentados en el elemento de red.

Debido a que un Protocolo de Administración de Red no está exento de cambios en la definición de comandos y servicios que deben de ser soportados por el Agente, se tiene como principal objetivo de diseño en el framework tener la capacidad suficiente de poder definir nuevos comandos SNMP o CMIP que pudieran ser definidos en el futuro.

Por último se tiene como objetivo también documentar de manera detallada el procedimiento o la metodología utilizada para poder diseñar este framework, definida en la siguiente sección ("*Sección 1.4 Metodologías*"). Al documentar este proceso, se obtiene un mayor entendimiento de las actividades y la inversión de tiempo implicados en el desarrollo de frameworks.

1.3 Restricciones

Las restricciones en el framework a desarrollar son las siguientes:

- El framework permitirá desarrollar Agentes de Administración de Red que estén basados en los modelos de información OSI o Internet, por lo cual no se garantizará el desarrollo de Agentes que no cumplan alguno de estos dos modelos.

- El framework solamente soportará los protocolos de administración de red SNMP y CMIP, quienes están definidos en la capa de aplicación tanto en el modelo de Internet como en el de OSI, por lo cual, no se diseñarán clases del framework que abstraigan la utilización de protocolos de mas bajo nivel.

- No obstante que no abstrajeran protocolos de más bajo nivel, las aplicaciones deben utilizar un protocolo de transporte para hacer llegar los mensajes entre el Agente y el Administrador, por lo cual el framework utilizará como protocolo de transporte a TCP/IP.

- Debido a la gran cantidad de Elementos de Red, no se diseñarán las librerías de clases concretas, las cuales pertenezcan a Agentes de algún Elemento de Red en específico, ya que dichas clases deberán ser provistas por el desarrollador de las aplicaciones. No obstante se desarrollará un prototipo que permita ver la aplicación del framework, en donde se desarrollarán clases concretas para el Ruteador Cisco 2501.

1.4 Metodologías

En el aspecto del desarrollo del framework, se utilizarán algunas de las fases sugeridas en el método de "*Ingeniería de Dominio Orientada a Objetos*" (IDOO) [Chan, 1998], [Coad, 1995], como lo son:

- *Análisis de Dominio Orientado a Objetos (ADOO).* Fase de desarrollo de un framework en el cual se determina las metas y objetivos del framework, su dominio, el dominio de la aplicación, los elementos que intervienen en el dominio del framework, sus conceptos, asociaciones y eventos (Véase el "*Capítulo 4. Análisis de Dominio Orientado a Objetos*".).

- *Diseño de Dominio Orientado a Objetos (DDOO).* Mediante esta etapa, las abstracciones obtenidas a partir del ADOO son mapeadas en clases, lo cual permite diseñar el framework (Véase el "*Capítulo 6. Diseño de Dominio Orientado a Objetos*".).

No obstante, en la presente tesis se propone manejar una fase denominada *"Análisis de Variabilidad en los Componentes del Dominio*" (AVCD), la cual tome como entrada las abstracciones realizadas sobre los conceptos definidos en el ADOO, para verificar las similitudes y variabilidades que existen en cada uno de éstos conceptos, y los agrupe en componentes de alto nivel. Esta fase se tomará como entrada en la fase de DDOO para modelar las variabilidades y similitudes de cada uno de los componentes identificados, lo cual es un elemento importante para diseñar el framework. La fase de AVCD se encuentra descrita en el "*Capítulo 5. Análisis de Variabilidad en los Componentes del Dominio*".

Por último, para el modelado de clases, objetos, escenarios, etc., que estén involucrados en el diseño del framework, se utilizará el Lenguaje Unificado de Modelación (UML).

1.5 Producto Final

Como producto final se tendrá el diseño de un framework orientado a objetos que sea la base para el desarrollo de Agentes de Administración de Red basados en el modelo OSI o en el modelo de Internet.

Además se tendrá un prototipo de aplicación del framework, el cual implementa a un Agente de Administración de Red para el Ruteador Cisco 2501, lo cual valida el diseño del framework.

1.6 Trabajos Relacionados

Un número considerable de frameworks exitosos ha sido desarrollado durante los últimos años [Hueni, 1995]. En el ámbito de la administración de redes, [Hueni, 1995] convierte un framework de protocolos de comunicación de caja blanca en un framework de caja negra.

[Kocher, 1999] diseña un framework para la arquitectura de control del sistema Alcatel 1641SX Digital Cross-Connect, el cual cumple con el rol de un Agente de Administración de Red, utilizando el protocolo CMIP.

No obstante que el framework propuesto por Kocher [Kocher, 1999] para Agentes de Administración de Red pudiera ser muy similar a lo propuesto por la presente tesis, el framework de Kocher está totalmente acoplado al modelo OSI y obviamente al protocolo CMIP. Además, sólo permite la ejecución de comandos CMIP definidos en [CCITT X.711, 1992], por lo cual si existe un cambio o una adición de comandos en el protocolo CMIP, el framework tendría que ser modificado para poder soportar éste cambio, lo cual no debe suceder en el framework propuesto por la presente tesis, ya que se tiene como objetivo poder extender los comandos del protocolo a de Administración de Red que el Agente pueda interpretar y ejecutar.

1.7 Contribución Esperada

La principal contribución que se espera de esta tesis, es la de describir a detalle el proceso de desarrollo de un framework en cada una de sus fases, aportando como valor agregado poder unificar dentro de un solo framework, tanto el modelo OSI como el modelo de Internet, los cuales son los modelos más utilizados en la actualidad para desarrollar aplicaciones de administración de red.

1.8 Estructura General de la Tesis

La estructura general de la tesis es la siguiente:

Capítulo 2 Agentes de Administración de Red. Introduce los procesos, comandos y acciones con los que un Agente de Administración de Red debe contar, tanto en el modelo OSI como en el de Internet.

Capítulo 3 Frameworks de Dominio Orientados a Objetos. Determina el proceso y la metodología que se debe seguir para desarrollar un framework de dominio.

Capítulo 4 Análisis de Dominio Orientado a Objetos. Se aplica la fase del ADOO al análisis, demarcación y ámbito del dominio de Agentes de Administración de Red, determinando los conceptos que intervienen en el mismo.

Capítulo 5 Análisis de Variabilidad en los Componentes del Dominio. Se definen las características comunes y de variabilidad de los conceptos de dominio determinados en la fase de ADOO.

Capítulo 6 Diseño de Dominio Orientado a Objetos. Las características comunes y de variabilidad de los conceptos de dominio, así como las problemáticas detectadas en la fase del AVCD son resueltas y mapeadas en clases, con lo cual se diseña el framework.

Capítulo 7 Implementación del Framework de Dominio. Se implementan las clases de la fase de DDOO, las cuales representan tanto la extensibilidad como la lógica de control del framework.

Capítulo 8 Aplicación del Framework de Dominio. Se presenta un ejemplo de aplicación del framework para un Agente de Administración de Red del Ruteador Cisco 2501.

Capítulo 9 Conclusiones Generales. Se presentan las conclusiones generales de cada una de las fases de desarrollo del framework así como los trabajos futuros que de la presente tesis se puedan derivar.

Anexo A Casos de Uso del Dominio. Se describen en este capítulo los casos de uso de dominio a diferentes niveles de abstracción, los reales, los abstractos y los del dominio.

Anexo B Diagramas Conceptuales de Colaboración del Dominio. Se presentan los diagramas conceptuales de colaboración correspondientes a los casos de uso del dominio.

Anexo C Descripción de Clases del Framework. Cada una de las clases del framework son descritas a través de sus miembros y métodos.

Anexo D Implementación del Agente Proxy SNMP para el Ruteador Cisco 2501. Se presenta la implementación de la aplicación Agente para el Ruteador Cisco 2501, presentando cada una de las clases que extienden del framework.

Capítulo

2

Agentes de Administración de Red

2.1 Introducción

A largo plazo, una red no es valiosa si no puede ser administrada propiamente. Se puede imaginar la dificultad de interconectar y comunicar diferentes máquinas, tales como computadoras, conmutadores, ruteadores, etc., si las convenciones difieren para administrar el uso de alarmas, indicadores de desempeño, estadísticas de tráfico, bitácoras, y otros elementos vitales de una red. La dificultad de administrar estos recursos se vuelve incrementalmente compleja cuando se añaden a las redes más y más componentes, más funciones y más usuarios. Debido a esta problemática es que surge el concepto de Administración de Red, la cual se encarga de planear, organizar, monitorear, contabilizar, y controlar las actividades y los recursos de una red [Black, 1992].

Dentro de la administración de redes, existen dos componentes fundamentales que interactúan entre sí para proporcionar los servicios que ella brinda, dichos componentes son los administradores y los agentes. Un administrador toma el papel principal en el monitoreo y control de agentes en su área de control. Los agentes, a su vez, interpretan los comandos enviados por los administradores y realizan las acciones correspondientes dependiendo de cada comando. Un agente se encarga también de reportar situaciones excepcionales a los administradores, tales como la pérdida de un enlace de datos, el desbordamiento de un buffer, etc. Los administradores y los agentes se comunican entre sí mediante intercambio de información, el cual se lleva a cabo a través de un protocolo de administración de red [Udupa, 1999]. La Figura 2.1 muestra la relación entre administradores y agentes.

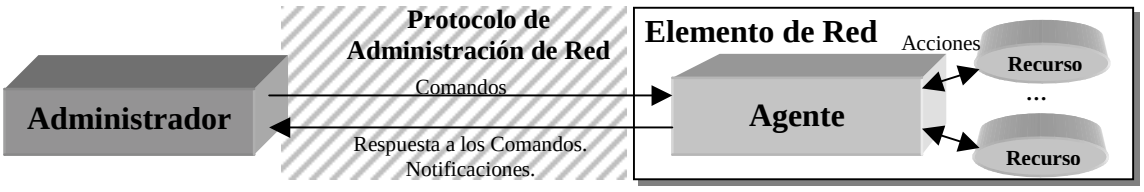


Figura 2.1 Relación entre Agentes y Administradores de Red.

La presente tesis observa a la administración de redes desde el punto de vista de un agente, por lo cual en este capítulo se abordan aquellos aspectos relevantes que tengan que ver con este elemento. Estos aspectos son muy importantes para esta tesis, debido a forman parte de los fundamentos que se tomarán en cuenta para desarrollar el framework.

Por último, cabe mencionar que todos los componentes que forman parte de los agentes se verán desde dos puntos de vista, tanto el de Internet como el de OSI, debido a que hoy en día son los estándares de administración de red más reconocidos en el ámbito mundial, además de que son también los más utilizados, por tanto, el framework a desarrollar dará soporte a estas dos arquitecturas.

2.2 Definición

Un Agente de Administración de Red es el responsable de reportar el estado actual de elementos de red a uno o varios Administradores. Se encarga también de recibir directivas por parte del Administrador, las cuales se derivan en acciones a desempeñar en estos elementos [Black, 1992].

Un agente reporta el estado de un elemento de red ya sea cuando el administrador así lo solicita (por medio de comandos), o cuando existe un recurso de este elemento en el cual se observa un comportamiento no deseado o fuera de ciertos parámetros ya establecidos (por ejemplo, la pérdida de conexión en una interfaz Ethernet de un ruteador).

Antes de profundizar en los diferentes componentes que forman parte de los Agentes de Administración de Red, conviene ubicar la funcionalidad y el trabajo que desempeñan estos Agentes en colaboración con otras entidades. Es por ello que la siguiente Sección brinda un panorama general de los elementos que forman parte integral de la administración de redes, así como un breve indicio de las funciones que desempeña un agente para cumplir con su papel.

2.3 Componentes de la Administración de Red

Los estándares de administración de red de OSI e Internet definen la responsabilidad que tiene tanto un Administrador como un Agente. En el estricto sentido, un sistema de administración de red contiene protocolos que transportan información sobre elementos de red de una ubicación a otra entre varios Agentes en el sistema y el Administrador [Black, 1992].

Existe otro componente que es vital para un sistema de administración de red. Se le denomina Base de Información de Administración (MIB de sus siglas en inglés). Este objeto conceptual es actualmente una base de datos que es compartida entre Administradores y Agentes para proveer información sobre los elementos de red administrados.

Ahora bien, ¿Donde residen todos estos elementos?. Los estándares de administración de red de OSI e Internet no requieren que sean ubicados en alguna localidad de la red en particular. No obstante, la Figura 2.2 presenta una configuración típica en una LAN la cual muestra la localización del Administrador, del Agente y de la MIB.

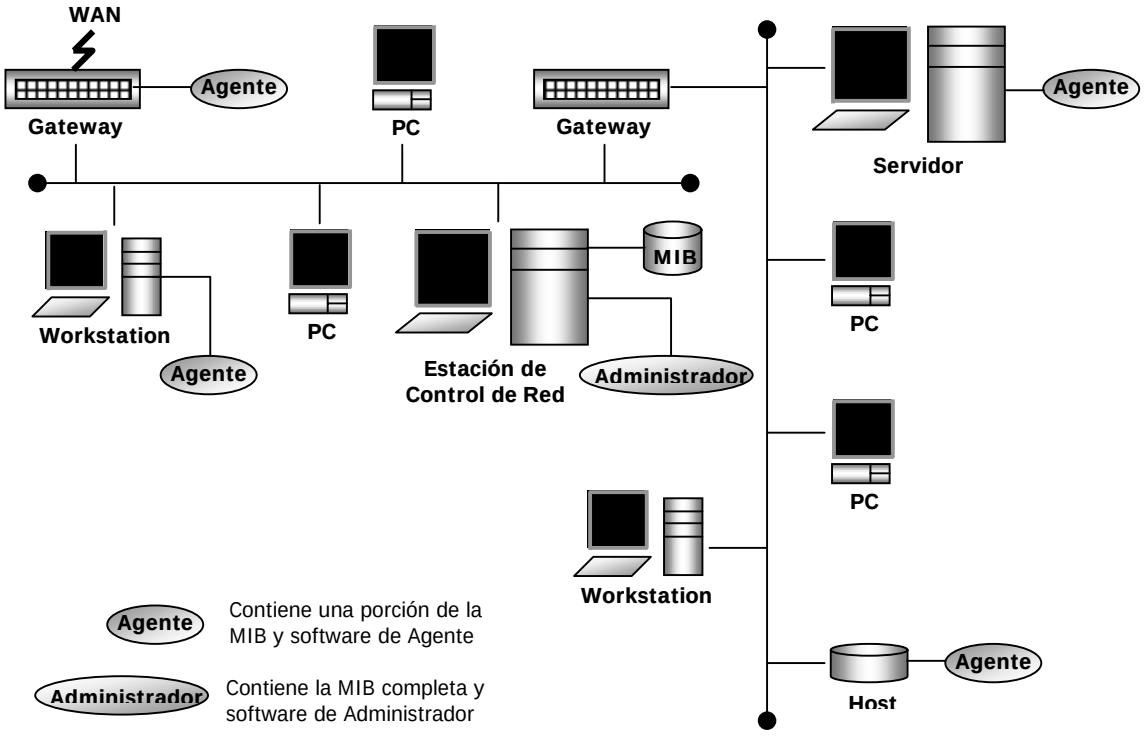


Figura 2.2 Localización de los Componentes de Administración de Red.

En implementaciones actuales, el software del agente es usualmente colocado en componentes tales como servidores, gateways, puentes y ruteadores. Típicamente una estación de control de red actúa como el proceso administrador. La MIB es usualmente localizada en la estación de control de red y la parte de la MIB que es pertinente al agente también es localizada en el agente.

En resumen, existen principalmente dos entidades con las que el agente interactua, el administrador y la MIB. Como ya se mencionó con anterioridad, tanto el agente como el administrador utilizan un "lenguaje común" o protocolo para poder colaborar entre sí y poder llevar a cabo sus funciones. Dada esta panorámica, en las siguientes Secciones se revisarán con más detalle todos los elementos que son utilizados por el agente, como lo son el protocolo de administración de red, la MIB y una entidad de la cual los diversos autores que tratan el tema de la administración de redes no mencionan comúnmente, la entidad de Administración Virtual de Hardware.

2.4 Protocolos de Administración de Red

Como ya se observó con anterioridad, las responsabilidades de un agente recaen en reportar al administrador el estado del elemento de red que esté bajo su cargo, además de la interpretación de comandos que deben de ser ejecutados en representación del administrador. Todo esto no puede llevarse a cabo si ambos no tienen en común una manera de comunicarse entre sí, y es por ello que es necesario un protocolo de administración de red que ambos deben compartir.

Debido a que en esta tesis se investigarán las arquitecturas propuestas por OSI e Internet, en esta Sección se describirán brevemente tanto el Protocolo Común de Información de Administración (CMIP, de sus siglas en inglés) como el Protocolo Simple de Administración de Red (SNMP, de sus siglas en inglés) respectivamente.

Cabe destacar que SNMP y CMIP pertenecen a la capa de aplicación, y por lo tanto, utilizan una pila de protocolos que están por debajo de esta capa. No obstante, la presente tesis no abarcará ningún aspecto que tenga que ver con algún protocolo que sea utilizado por SNMP o por CMIP, debido a que el framework que se pretende diseñar no pretende abstraer el comportamiento de protocolos que estén por debajo de este nivel.

Cada uno de estos protocolos utiliza un esquema de solicitud y respuesta muy comúnmente utilizado en los sistemas cliente/servidor. Cada solicitud y respuesta contiene una serie de parámetros utilizados para la comunicación entre el administrador y el agente. No obstante que en las siguientes secciones se presentarán los diferentes mensajes (comandos, solicitudes y respuestas) que son permitidos para cada protocolo, no se describirá a fondo el formato de cada uno de los parámetros que son utilizados en los mensajes, pero se puede encontrar más información en los estándares que definen a cada protocolo ([RFC-1157, 1990], [CCITT X.710, 1992]).

2.4.1 Protocolo Común de Información de Administración (CMIP)

El estándar ISO 9595 [CCITT X.710, 1992] establece la especificación para el protocolo CMIP. CMIP soporta los servicios que a continuación se mencionan:

M-EVENT-REPORT. Es utilizado para reportar eventos o condiciones excepcionales. De esta manera, el servicio M-EVENT-REPORT es utilizado para soportar la función de reporte de alarmas de parte de un agente a un administrador. Este servicio puede ser utilizado tanto en modo confirmado o no confirmado[3]. La Figura 2.3 muestra la secuencia de información utilizada para reportar un evento por parte del agente a su administrador, así como el significado de cada uno de sus datos.

[3] En Figuras posteriores, no se esquematizará el modo no confirmado, debido a que es similar al modo confirmado, con excepción de la respuesta que se obtiene al solicitar el servicio.

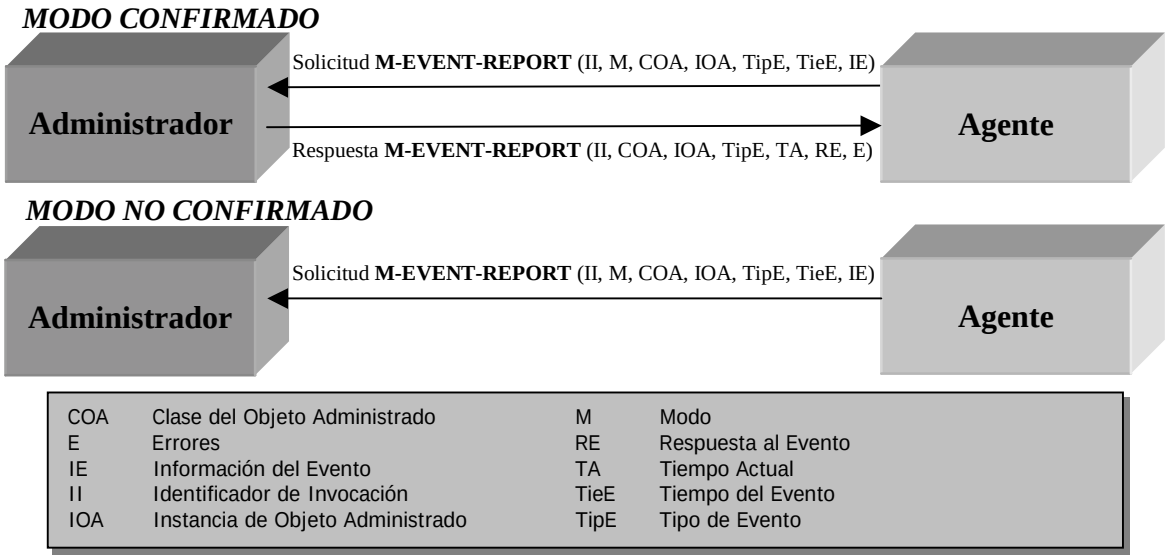


Figura 2.3 Secuencia de Información del Servicio M-EVENT-REPORT

M-GET. Es utilizado para obtener información de uno o varios atributos de los objetos administrados que están a cargo del agente. Típicamente este servicio es utilizado para obtener indicadores de estado en objetos administrados en la red o información sobre la eliminación o adición previa de objetos. Este es un servicio confirmado. La Figura 2.4 nos muestra la secuencia de información utilizada por el servicio GET, así como el significado de cada uno de sus datos.

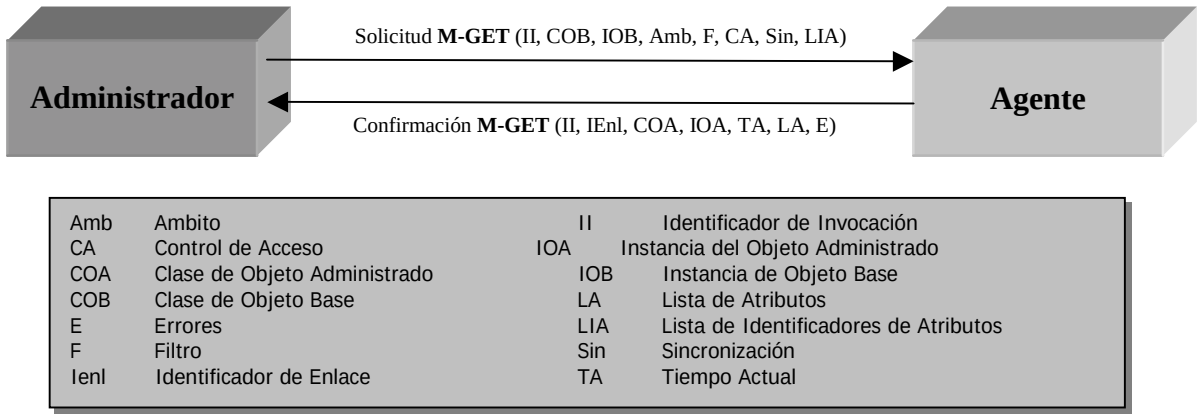


Figura 2.4 Secuencia de Información del Servicio M-GET

M-CANCEL-GET. El servicio GET es el único que tiene una opción de cancelación. Es utilizado para cancelar la solicitud de la información de uno o varios objetos administrados. Este servicio es del tipo confirmado. La Figura 2.5 ilustra las operaciones M-CANCEL-GET y los parámetros asociados con este servicio.

MODO CONFIRMADO

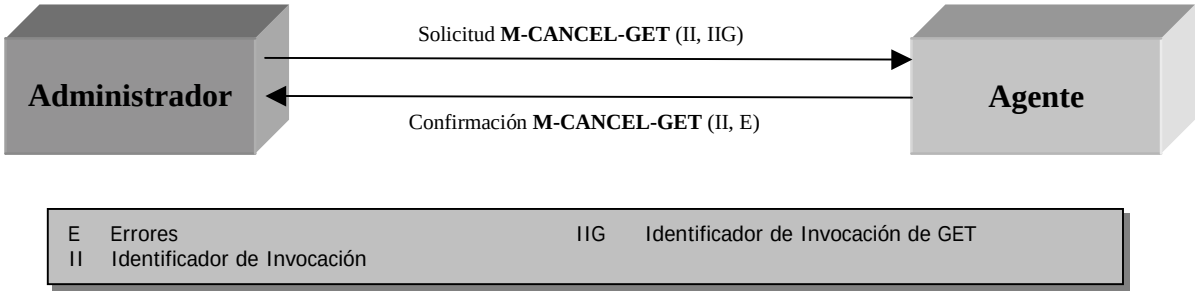


Figura 2.5 Secuencia de Información del Servicio M-CANCEL-GET

M-SET. Este servicio es utilizado para solicitar el cambio de valores de atributos en un objeto administrado. La Figura 2.6 ilustra las operaciones de M-SET y los parámetros asociados con este servicio.

MODO CONFIRMADO

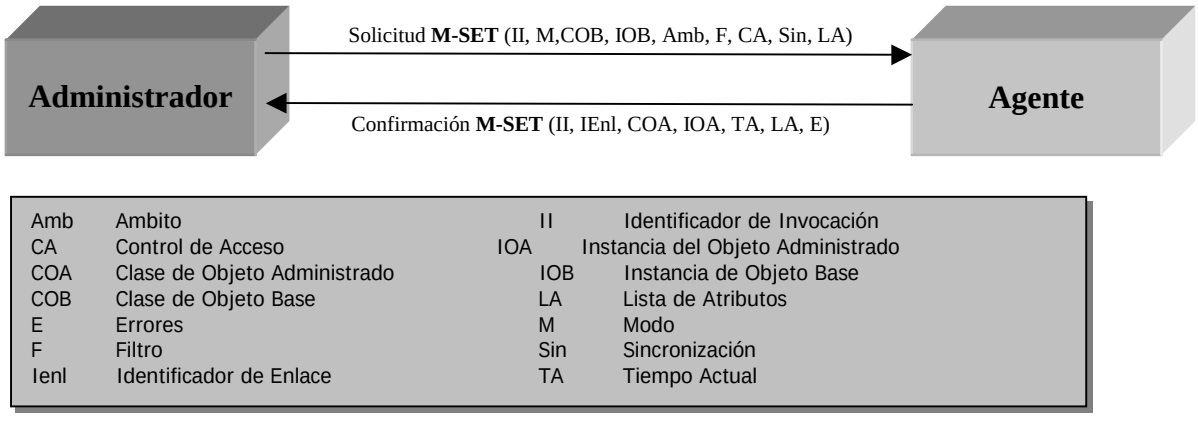


Figura 2.6 Secuencia de Información del Servicio M-SET

M-ACTION. Este servicio es utilizado para la invocación de un servicio a operar en un objeto administrado, es decir, desempeñar una acción en un objeto administrado. Puede ser utilizado como un servicio confirmado o no confirmado. La Figura 2.7 ilustra las operaciones de M-ACTION y los parámetros asociados con este servicio.

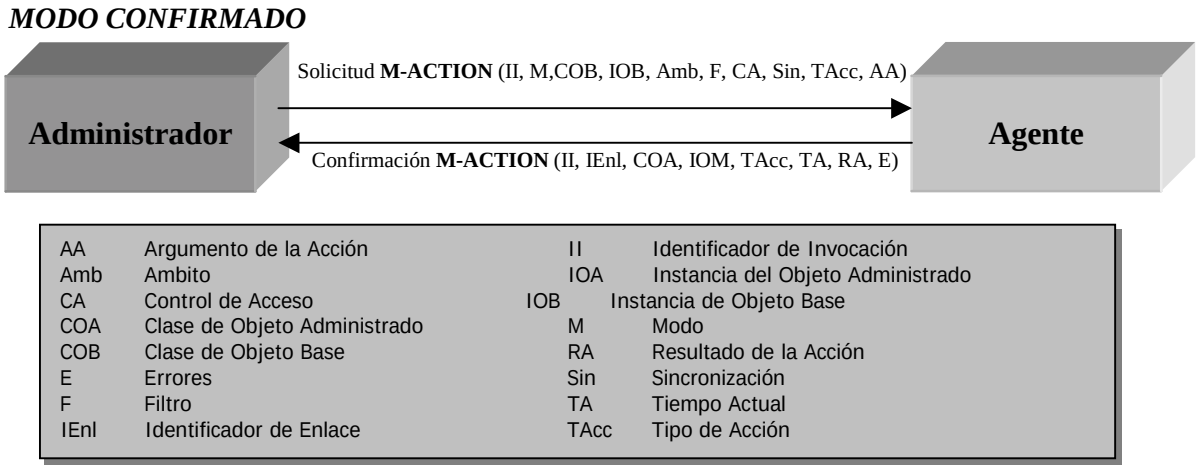


Figura 2.7 Secuencia de Información del Servicio M-ACTION

M-CREATE. Este servicio es solicitado para crear la representación de una nueva instancia de un objeto administrado. Este servicio solo puede ser utilizado en modo confirmado. La Figura 2.8 ilustra las operaciones de M-CREATE y los parámetros asociados con este servicio.

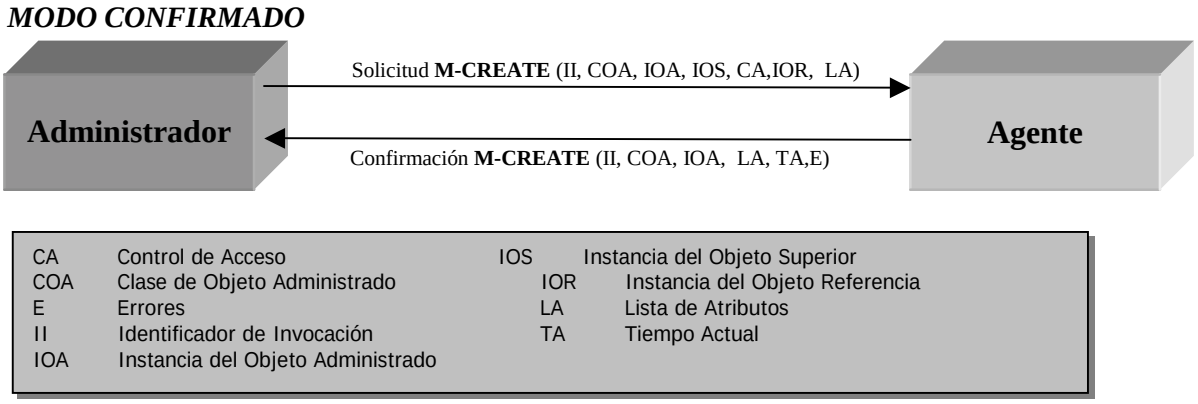


Figura 2.8 Secuencia de Información del Servicio M-CREATE

M-DELETE. Este servicio es utilizado para borrar una representación de una instancia de un objeto administrado. Solamente es utilizado en modo confirmado. En la Figura 2.9 se muestran las operaciones de M-DELETE y los parámetros asociados con este servicio.

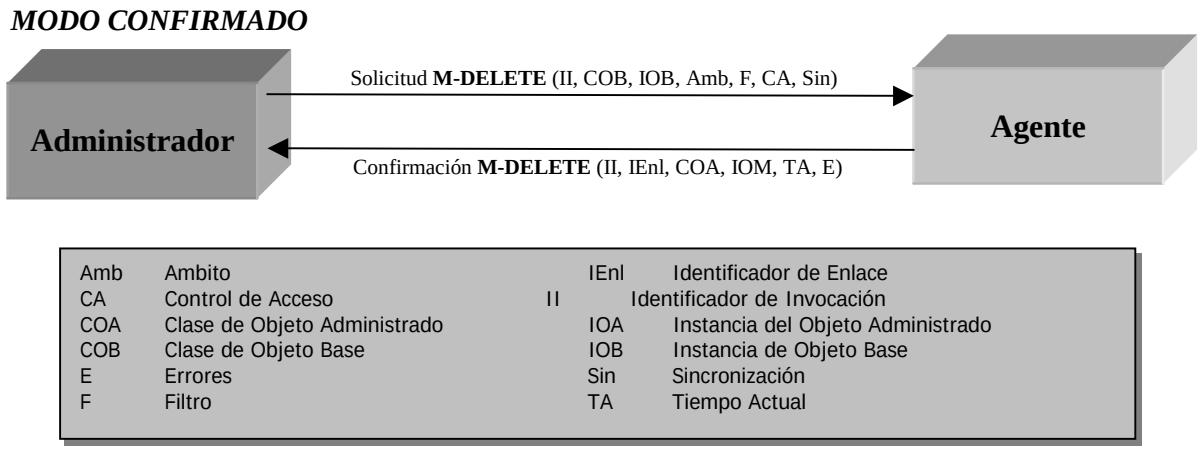


Figura 2.9 Secuencia de Información del Servicio M-DELETE

Como otros protocolos OSI, CMIP debe seguir reglas de composición e intercambio de Unidades de Datos de Protocolo (PDUs). Todas las PDUs de CMIP son definidas conforme a la Notación de Sintaxis Abstracta versión 1 (ASN.1).

2.4.2 Protocolo Simple de Administración de Red (SNMP)

El estándar de administración de red más utilizado es el Protocolo Simple de Administración de Red (SNMP). SNMP opera con dos funciones de administración:

1. Una entidad interactua con un agente para obtener variables.
2. Una entidad interactua con un agente para modificar los valores de estas variables.

La idea detrás de esto es una simple metodología que limite significantemente las funciones que pueden ser desempeñadas con SNMP, lo cual, como una consecuencia, limita la complejidad del software.

Estas funciones pueden ser implementadas a través de operaciones "polling", es decir, un administrador SNMP puede ser programado para enviar periódicamente mensajes a los dispositivos administrados a ciertos intervalos. Estos intervalos pueden ser establecidos a través de la MIB de SNMP.

SNMP no es totalmente un protocolo "polling". Permite un tipo de tráfico no solicitado llamado "traps", basado en parámetros restringidos. En este caso, el agente es responsable de desempeñar chequeos[4] y reportar solamente condiciones que pertenezcan a cierto criterio, por ejemplo, un indicador de temperatura que llegó a ser muy caliente, etc. La Figura 2.10 ilustra los dos tipos de funciones de administración antes vistas.

[4] Comúnmente llamados filtros

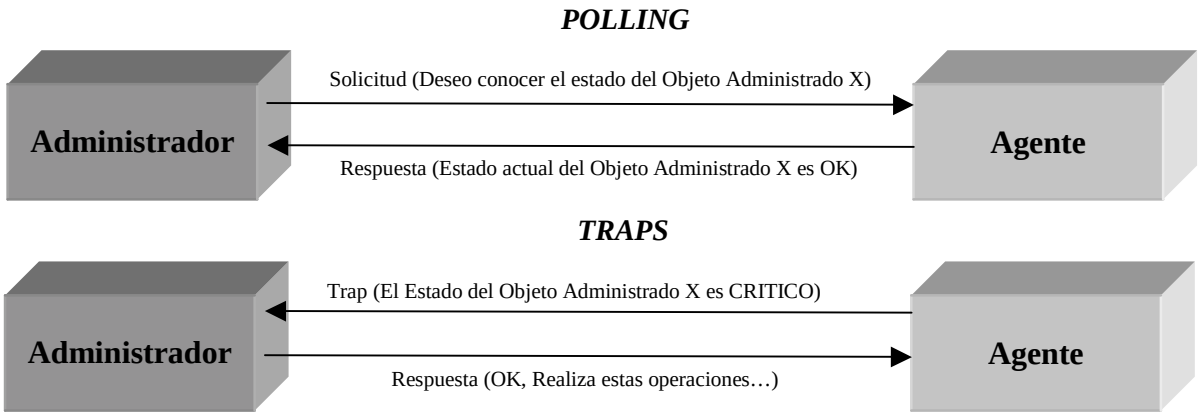


Figura 2.10 Funciones de Administración de SNMP

Este protocolo utiliza operaciones relativamente simples y un número limitado de PDUs para desempeñar sus funciones. Cinco PDUs han sido definidos en el estándar[5]. Estos son:

GET-REQUEST. Este PDU es utilizado para accesar el agente y obtener valores de una lista. Contiene identificadores para distinguirse de múltiples solicitudes, así como también valores para proveer información sobre el estado del elemento de red. La Figura 2.11 muestra la secuencia de información del PDU así como los parámetros asociados con él.

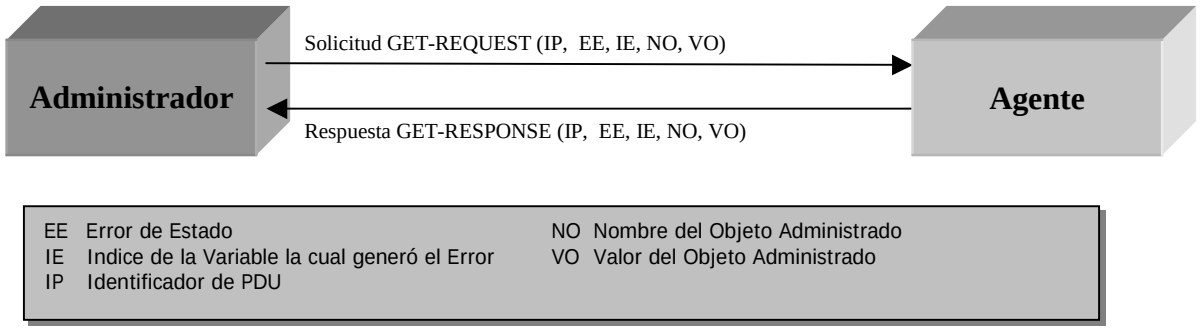


Figura 2.11 Secuencia de Información del Servicio GET-REQUEST

GET-NEXT-REQUEST. Este PDU es similar a GET-REQUEST, excepto que permite la obtención del siguiente identificador lógico en un árbol MIB. La Figura 2.12 muestra la secuencia de información de este PDU así como los parámetros asociados con él.

[5] Cabe destacar que ninguno de estos PDUs utiliza el modo de confirmación utilizado por CMIP.

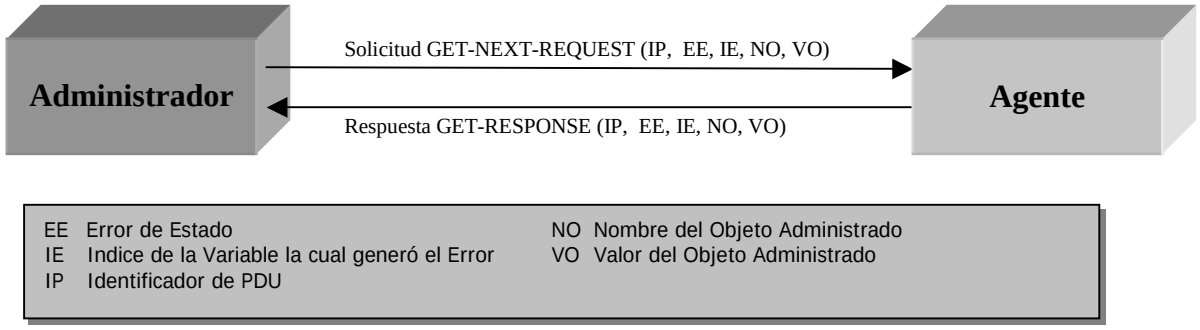


Figura 2.12 Secuencia de Información del Servicio GET-NEXT-REQUEST

SET-REQUEST. Es utilizado para describir una acción a ser desempeñada en un elemento. Típicamente, es usada para cambiar los valores en una lista de variables. La Figura 2.13 muestra la secuencia de información de este PDU así como los parámetros asociados con él.

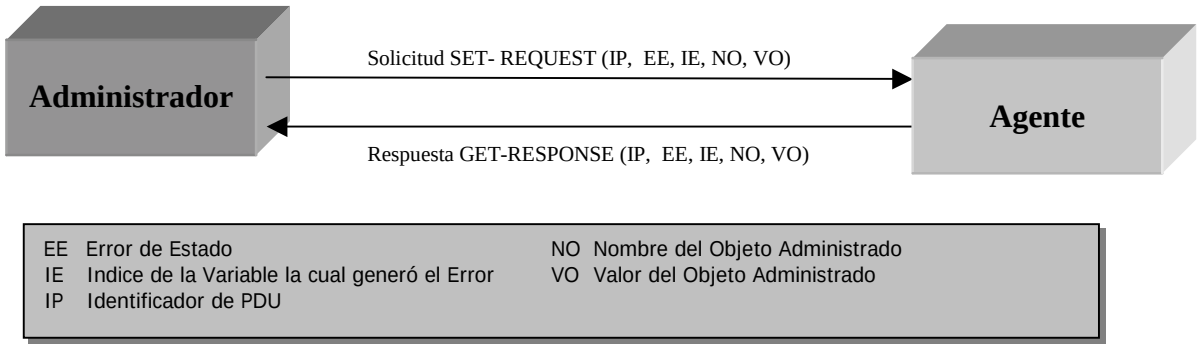


Figura 2.13 Secuencia de Información del Servicio SET-REQUEST

GET-RESPONSE. Este PDU responde a las PDUs GET-REQUEST, GET-NEXT-REQUEST, y SET-REQUEST. Contiene un identificador que se asocia con un identificador de un PDU anterior. También contiene identificadores para proveer información sobre el estado de la respuesta (códigos de error, error de estado, y una lista de información adicional). Cualquiera de las Figuras 2.11 a 2.13 muestra la información que contiene este PDU.

TRAP. Este PDU permite al agente reportar sobre un evento en un elemento de red o cambiar el estado del elemento de red. La Figura 2.14 muestra la información que contiene el presente PDU.

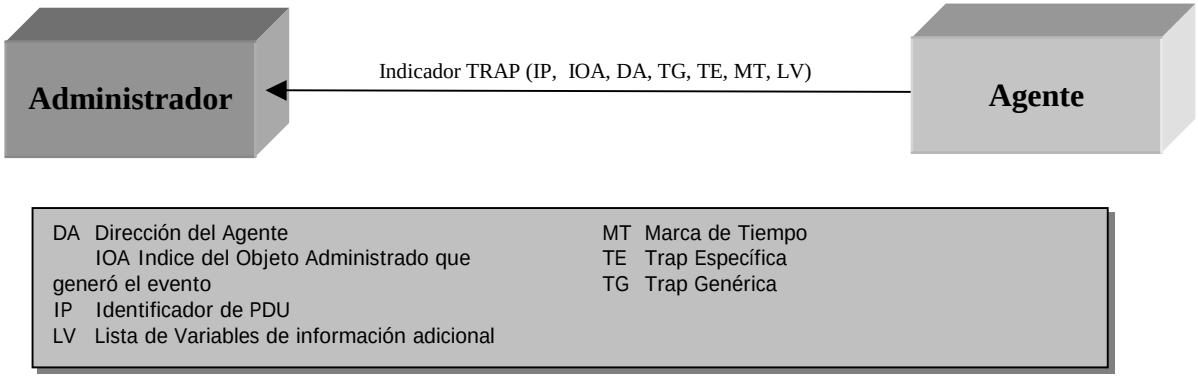


Figura 2.14 Secuencia de Información del Servicio TRAP

Ahora se ha revisado cada uno de los comandos de los protocolos de administración de red CMIP y SNMP, los cuales dará soporte el framework a desarrollar, conviene verificar también el elemento fundamental que permite preservar la información de administración tanto en los agentes como en los administradores. A esta entidad se le denomina Base de Información de Administración (MIB).

2.5 Base de Información de Administración (MIB)

La Base de Información de Administración (MIB) consiste de un conjunto de objetos y sus atributos. La idea de la MIB es el enlazar sistemas de administración, junto con operaciones correspondientes a las diferentes capas de cada arquitectura (OSI o Internet).

Las MIBs forman el componente principal de la administración de redes. Es en la MIB donde se definen los objetos administrados ya que contiene sus nombres, su comportamiento permisible y las operaciones que pueden ser desempeñadas sobre ellos.

Dentro de las MIBs existen recursos que son supervisados y controlados por el Administrador los cuales son llamados Objetos Administrados. Un Objeto Administrado puede ser cualquier elemento importante a las organizaciones que utilizan ya sea los estándares OSI o de Internet. Como ejemplo tenemos a hardware como conmutadores, ruteadores, estaciones de trabajo, PBXs, puertos de tarjeta PBX, y multiplexores, todos ellos pueden ser identificados como objetos administrados.

2.5.1 Objetos Administrados de OSI

Desde la perspectiva de OSI, un objeto administrado es descrito y definido por cuatro aspectos de la administración de redes:

- **Atributos**. Los objetos administrados tienen ciertas propiedades que los distinguen de otros. Estas propiedades son llamadas atributos. El propósito de un atributo es describir las características, estado actual, y condiciones de la operación de los objetos administrados.
- **Operaciones**. Las operaciones permisibles sobre los objetos administrados son prácticamente las descritas en la Sección correspondiente al protocolo CMIP (M-CREATE, M-GET, etc.)
- **Notificaciones**. A los objetos administrados se les permite enviar reportes sobre eventos que ocurren en él.
- **Comportamiento**. Un objeto administrado puede exhibir características de comportamiento. Estas incluyen tanto como el objeto reacciona a las operaciones desempeñadas en él y restricciones que son impuestas en su comportamiento.

2.5.2 Objetos Administrados de Internet

Desde la perspectiva de Internet, un objeto administrado es descrito de una manera menos abstracta que la correspondiente a OSI. Un objeto administrado es descrito por:

- **Sintaxis**. La sintaxis define el tipo de dato. En contraste con la administración de red de OSI, los tipos de datos de Internet son bastante limitados.
- **Acceso**. La definición del acceso provee información sobre como un objeto administrado puede ser accesado respecto a si es de solo lectura, escritura, etc.
- **Estado**. El estado de un objeto es definido actualmente como obligatorio, opcional u obsoleto. El estado nos indica si cierto nodo es indispensable o no para implementar su objeto administrado.
- **Nombre**. La definición del nombre es un identificador de objeto, el cual es utilizado de manera no ambigua para identificar un objeto administrado.

2.5.3 OSI en contraste con Internet

El modelo de Internet no hace distinción entre objetos y atributos. Consecuentemente, el reuso de atributos para otros objetos no son permitidos. Esto quiere decir que el modelo de OSI utiliza de una manera más apropiada y extensa los conceptos de orientación a objetos.

Desde una perspectiva de OSI, los objetos administrados que tienen similares características son agrupados dentro de una clase de objeto, la cual es llamada una clase de objeto administrado. Las clases de objetos administrados proveen un conveniente significado para agrupar recursos, y un nombrado jerárquico, además de que se permite el derivar nuevas clases de las ya existentes, utilizando el mecanismo de herencia.

El modelo de Internet no trata con clases de objetos de la misma forma que el modelo OSI, la aproximación más cercana a una clase de objeto es el llamado grupo. Para propósitos de clasificación, Internet agrupa objetos administrados en grupos. Por ejemplo, el grupo de interfaces incluye a todos los objetos que pertenecen a una capa física de enlace.

No obstante que tanto Internet y OSI manejan el concepto de objeto administrado de distinta manera[6], existe un aspecto que las dos arquitecturas tienen en común. Ambas manejan un Árbol de Información de Administración[7] (MIT de sus siglas en inglés), el cual es utilizado como el fundamento para la administración e identificación de recursos (objetos administrados) en una red.

2.6 Administración Virtual de Hardware

Por debajo de la MIB, existe una capa de abstracción de hardware, la cual es denominada como la Administración Virtual de Hardware(AVH). Esta capa ofrece una interfaz lógica para todos los recursos de hardware que es independiente de cualquier modelo de información específico [Fayad 2, 1999]. La Figura 2.15 nos muestra la posición de la AVH respecto a las demás capas que están implícitas en cada uno de los elementos de red que son administrados.

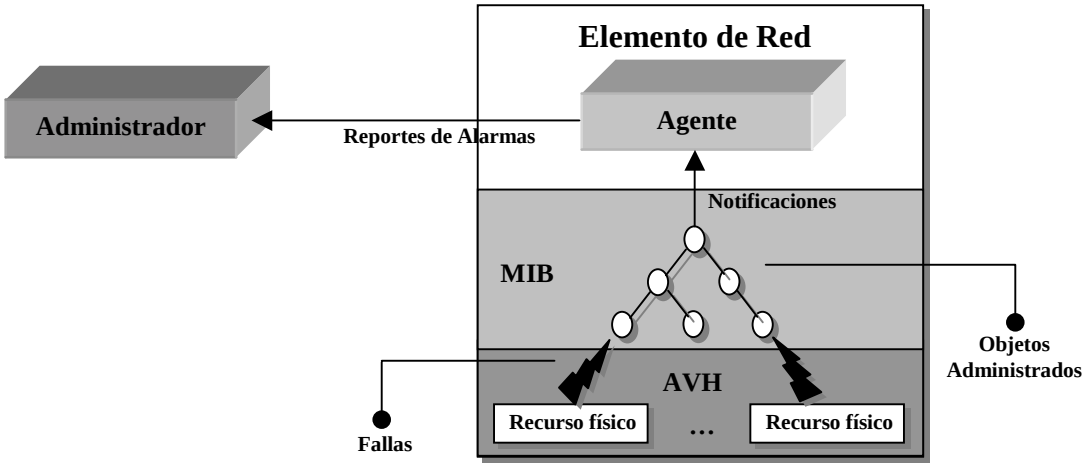


Figura 2.15 Posición de la AVH respecto a las demás capas.

La AVH transforma las solicitudes lógicas de la capa de MIB a solicitudes específicas de hardware. También es responsable de la detección de defectos en las señales de transmisión y de componentes de hardware. Las alarmas correspondientes son reportadas a la MIB.

---

[6] Desde un sentido práctico, los objetos administrados de Internet se asemejan a los atributos de OSI.

[7] Se le llama de igual manera en la arquitectura OSI, y en la de Internet se le denomina Sistema de Nombres de Dominio.

2.7 Proceso de Reporte de Alarmas de los Modelos de Información OSI e Internet

Existe una función primordial a la cual el Agente debe dar soporte y ésta es el reporte de alarmas que son originadas por un recurso que le pertenezca. Las alarmas son situaciones excepcionales que pueden ocurrir en recursos que están en el ámbito de vigilancia del Agente, como lo puede ser el desbordamiento de un buffer, la desconexión de una interfaz Ethernet, etc. Es por ello que en esta Sección se trata este tema, tanto para la arquitectura OSI como para la de Internet.

Cabe mencionar que el proceso de reporte de alarmas comienza cuando la administración virtual de hardware (la cual como ya se mencionó es independiente de cualquier modelo de información específico) detecta un cambio de estado en uno o varios de los recursos de un elemento de red. En este caso se le reporta a la MIB de este cambio de estado. La MIB asocia entonces estos recursos con sus correspondientes objetos administrados, los cuales tienen la tarea de actualizar sus atributos, y dependiendo del modelo de información (OSI o Internet), estos reportan a su vez al agente para su posterior análisis y envío de alarmas a su administrador. No obstante que este es un escenario muy común en los dos modelos de información, es necesario analizar con mayor profundidad las diferentes actividades llevadas a cabo tanto por OSI como por Internet, las cuales se describen a continuación.

2.7.1 Proceso OSI de Reporte de Alarmas

La Supervisión de Alarmas es el proceso de recepción y organización de alarmas el cual se realiza previamente al procesamiento realizado por el administrador.

Un componente clave de la Supervisión de Alarmas es el control ejercido sobre el reporte de alarmas y el monitoreo del estado de alarmas excepcionales. Esto es importante dado que el administrador necesita conocer rápida y exactamente cuando las alarmas ocurren y también cuando la alarma es dilucidada. La Función de Reporte de Alarmas junto con la función de Reporte de Eventos describen el mecanismo empleado para el reporte de alarmas.

Frecuentemente es importante poner en bitácora las actividades de reporte de alarmas. Por ejemplo pudiera ser importante el mantener un seguimiento de las alarmas, el cual pueda servir para un análisis estadístico. La Función de Control de Bitácora describe los mecanismos utilizados para el almacenamiento en bitácora de los reportes de alarmas.

La Función de Reporte de Alarmas y la Función de Control de Bitácora emplean notificaciones que son utilizadas por los objetos administrados para generar reportes no solicitados. Los tipos de notificación que pueden ser producidos por un objeto administrado dependen de la definición de su clase de objeto administrado. Existen varios tipos posibles de notificaciones algunas de las cuales son tipos de alarmas.

En general, un objeto administrado generará una notificación de alarma cuando algo le ocurre al objeto administrado o al recurso asociado con el objeto administrado que necesita ser reportado al administrador, lo cual no fue el resultado de una operación solicitada por el administrador. Típicamente las alarmas ocurren cuando una falla o problema es detectado por los recursos.

A fin de manejar las notificaciones, varios objetos administrados de soporte pueden ser empleados. Los objetos administrados de soporte más comunes son la Bitácora y el Discriminador de Envío de Eventos. La Bitácora es un contenedor que procesa las notificaciones que representan las alarmas y las almacena como Registros de Bitácora. El Discriminador de Envío de Eventos es utilizado para enviar las alarmas en la forma de reporte de eventos al administrador.

La Figura 2.16 ilustra como las notificaciones son procesadas en el sistema administrador para generar Registros de Bitácora y Reportes de Eventos.

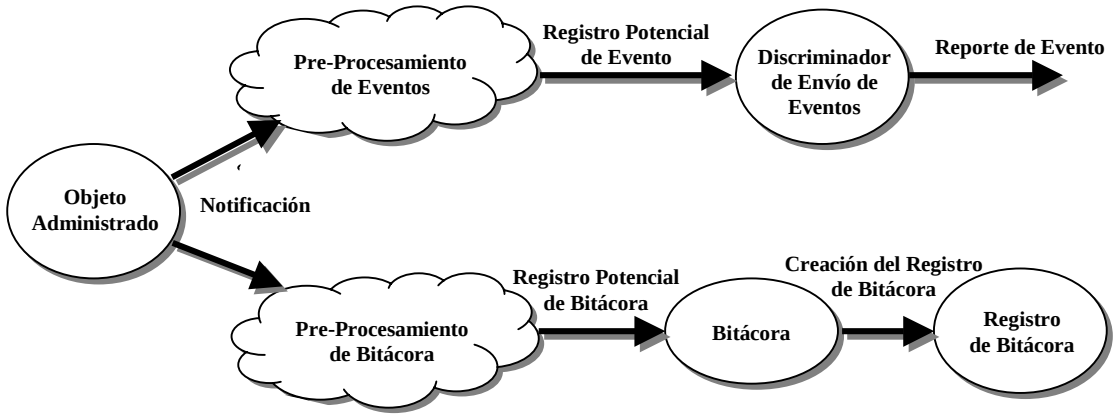


Figura 2.16 Notificaciones, Registros de Bitácora y Reporte de Eventos.

Habiendo producido una notificación que representa una alarma, el agente es responsable de asegurarse de que la notificación es enviada a las Bitácoras y Discriminadores de Envío de Eventos correctos para su procesamiento. Los estándares relevantes, la Función de Reporte de Eventos y la Función de Control de Bitácora, especifican que "conceptualmente" todas las notificaciones son enviadas a todas las Bitácoras y Discriminadores de Envío de Eventos. La Bitácora y el Discriminador de Envío de Eventos tienen la habilidad de filtrar las notificaciones que necesitan de todas las notificaciones en el sistema. Por ejemplo, en el caso de la supervisión de alarmas, solamente nos interesarán aquellas notificaciones que sean un tipo de alarma.

La acción que es tomada una vez que la alarma ha alcanzado a una Bitácora o un Discriminador de Envío de Eventos y ha sido seleccionada para un procesamiento posterior es diferente para la Bitácora y el Discriminador de Envío de Eventos.

La Bitácora toma la notificación entrante (registro potencial de Bitácora) seleccionada por el filtro y la convierte en un Registro de Bitácora, el cual es un objeto administrado. Este Registro de Bitácora es nombrado por la Bitácora y es contenido en ella.

El Discriminador de Envío de Eventos toma la notificación entrante (registro potencial de evento) seleccionado por el filtro y lo convierte en un reporte de evento que es enviado al administrador utilizando la operación de reporte de eventos que está determinada por el protocolo de administración de red utilizado.

El reporte de alarmas es una especialización del reporte general de eventos y el control de bitácora. Sin embargo las notificaciones de alarmas tienen varios campos de información adicional que son significantes y que proveen información valiosa acerca de la alarma y como está siendo tratada. Los campos más significantes son Causa Probable, Severidad Percibida y Problemas Específicos.

La Causa Probable califica el tipo de evento de la alarma indicando más precisamente la causa del problema que ha resultado en el reporte de dicha alarma. El parámetro de Problema Específico califica adicionalmente el tipo de alarma añadiendo mas detalles.

El parámetro de la Severidad Percibida tiene seis valores que indican la severidad del problema que causó las alarmas. Estos valores son usados en el parámetro de Severidad Percibida de la alarma cuando esta ocurre y en el Estado de la Alarma del objeto administrado que emitió la alarma. El estado de la alarma del objeto administrado puede cambiar de estado, o la alarma puede ser dilucidada completamente y esto resultará en una alarma que indica la transición. Típicamente y en muchos casos, un objeto administrado generará dos alarmas asociadas con cada problema: una alarma será generada cuando el problema ocurre; y otra alarma será generada cuando el problema es dilucidado. Los seis valores que la Severidad Percibida puede tomar son:

- Dilucidada.
- Indeterminada.
- Precaución.
- Menor.
- Mayor.
- Crítica.

Ante esto, se necesita un mecanismo que permita la asignación correcta de todos estos parámetros, es por ello que la Función de Perfil de Asignación de la Severidad de Alarmas permite al administrador el definir como los valores de Severidad son asignados a las alarmas por el sistema administrador.

Un objeto administrado que contiene un puntero a un Perfil de Asignación de la Severidad de Alarmas genera los valores específicos de Severidad que son asociados con alarmas. Varios objetos administrados pueden ser asociados con un Perfil de Asignación de Severidad de Alarmas.

El Perfil de Asignación de Severidad de Alarmas asocia los valores de Severidad con las Causas Probables. Los valores de Severidad son asignados a la Severidad Percibida de las alarmas cuando la alarma ocurre con esa Probable Causa, y el valor es también asignado al Estado de la Alarma del objeto administrado si es apropiado. Cada Perfil de Asignación de Severidad de Alarmas contiene una lista que relaciona Causas Probables con Severidad, esta lista puede ser cambiada por el administrador dependiendo de los cambios en los requerimientos de administración. Pueden existir varias instancias del Perfil de Asignación de Severidad de Alarmas presentes en el agente.

2.7.2 Proceso de Internet para el Reporte de Alarmas

En Internet se pretende que cada uno de los agentes y MIBs sean lo más simple posible en cuanto a capacidad de procesamiento y almacenamiento para llevar a cabo sus funciones de administración. Es por ello que el número de actividades realizadas por un agente de Internet para reportar alarmas, en comparación con un agente OSI, es relativamente más pequeño.

En Internet no se almacenan las notificaciones realizadas por los objetos administrados, es decir, no es necesario contar cono ningún tipo de bitácora, tampoco es necesario contar con un discriminador de envío de eventos, esto hace que el proceso de reporte de alarmas sea mucho más simple.

Cuando un objeto administrado cambia de estado (o de valor), este le reporta a su agente de dicho cambio. El agente a su vez desempeña un chequeo de los límites frontera (comúnmente llamado un filtro) los cuales en situaciones normales no debe sobrepasar el objeto administrado. En caso de que el estado del objeto haya sobrepasado dichos límites, el agente reporta al administrador toda la información pertinente mediante uno o varios traps, los cuales son enviados utilizando el protocolo SNMP.

2.8 Conclusiones

En el presente capítulo se tienen las siguientes conclusiones:

- Conocer los procesos que son llevados a cabo tanto por el modelo de información OSI como el de Internet, nos puede llevar a obtener las abstracciones necesarias para desarrollar el framework.

- El proceso llevado a cabo para reportar alarmas utilizando el modelo de información OSI, es mucho más complejo en comparación con el de Internet, es por ello que se recomienda implementar el modelo OSI en equipos sumamente grandes y con capacidad de procesamiento y almacenamiento considerable.

- No obstante la diferencia de complejidad que se presenta en los dos modelos, existen varias similitudes entre cada uno de los componentes (Agente, MIB, y la AVH), las cuales pueden resultar en abstracciones que permitan diseñar el núcleo del framework.

Capítulo

3

Frameworks de Dominio Orientados a Objetos

3.1 Introducción

Los frameworks de dominio orientados a objetos son aplicaciones que permiten abstraer dominios específicos, tales como interfaces de usuario, redes y telecomunicaciones, o ambientes de trabajo colaborativo basado en multimedia [Fayad, 1999]. Un framework es mas que una jerarquía de clases [Lewis, 1995]. Es una aplicación semicompleta conteniendo componentes estáticos y dinámicos que pueden ser adaptados para producir aplicaciones de usuario específicas [Fayad 3, 1999]. Debido a la naturaleza genérica de los componentes del framework, frameworks "maduros" pueden ser reutilizados como la base para muchas otras aplicaciones [Fayad 3, 1999].

Debido a que en la presente tesis se desarrolla un framework para Agentes de Administración de Red, es necesario tener un marco teórico sobre frameworks de dominio, el cual permita identificar los elementos que deben desarrollarse para llegar a implementar el framework y sus aplicaciones. Por lo cual, en este capítulo se presenta un panorama general de los conceptos que el desarrollo de frameworks involucra, su clasificación tanto en su ámbito como en su forma de extensión, la manera en que se pueden utilizar y la metodología de desarrollo a seguir en la tesis, la cual se irá viendo con detalle en cada uno de los capítulos posteriores, por tanto este capítulo solo es introductorio.

3.2 Definición

Una definición frecuentemente utilizada es "un framework es un diseño reusable de todo o parte de un sistema que es representado por un conjunto de clases abstractas y la manera en que sus instancias interactúan" Otra definición común es "un framework es un esqueleto de una aplicación que puede ser ajustada por un desarrollador de aplicaciones." Estas no son definiciones conflictivas; la primera describe la estructura de un framework mientras que la segunda describe su propósito. No obstante, ambas señalan la dificultad de definir a los frameworks claramente [Fayad, 1999].

Se puede decir también que un framework es una aplicación semicompleta reusable que puede ser especializada para producir aplicaciones. Es también un diseño reusable de un dominio que describe como el dominio es descompuesto en un conjunto de objetos interactuantes. El framework describe los objetos que lo componen y como éstos objetos interactúan. Describe la interfaz de cada objeto y el flujo de control entre ellos, es decir, describe como las responsabilidades inherentes en el dominio son mapeadas dentro de objetos [Johnson-Foote, 1988].

Típicamente, un framework es implementado con un lenguaje de programación orientado a objetos como C++, Smalltalk, y más recientemente Java. Cada objeto en el framework es descrito por una clase abstracta. Una clase abstracta es una clase que no puede ser instanciada, por tanto, es utilizada solamente como una superclase [Wirfs-Brock, 1990]. Una clase abstracta usualmente tiene al menos una operación no implementada delegada a sus subclases. Puesto que una clase abstracta no tiene instancias, es utilizada como una plantilla para crear subclases mas que una plantilla para crear objetos. Los frameworks las utilizan como diseños de sus componentes porque definen la interfaz de los componentes y proveen un esqueleto que puede ser extendido para implementar los componentes, a esto se le denomina como "*Extensibilidad del Framework*". No obstante, el framework debe tener implementado al menos un método que controle la ejecución de las aplicaciones, el cual se le denomina como "*Lógica de Control del Framework*".

Como ejemplo, la Figura 3.1 nos muestra parte de la estructura de un framework.

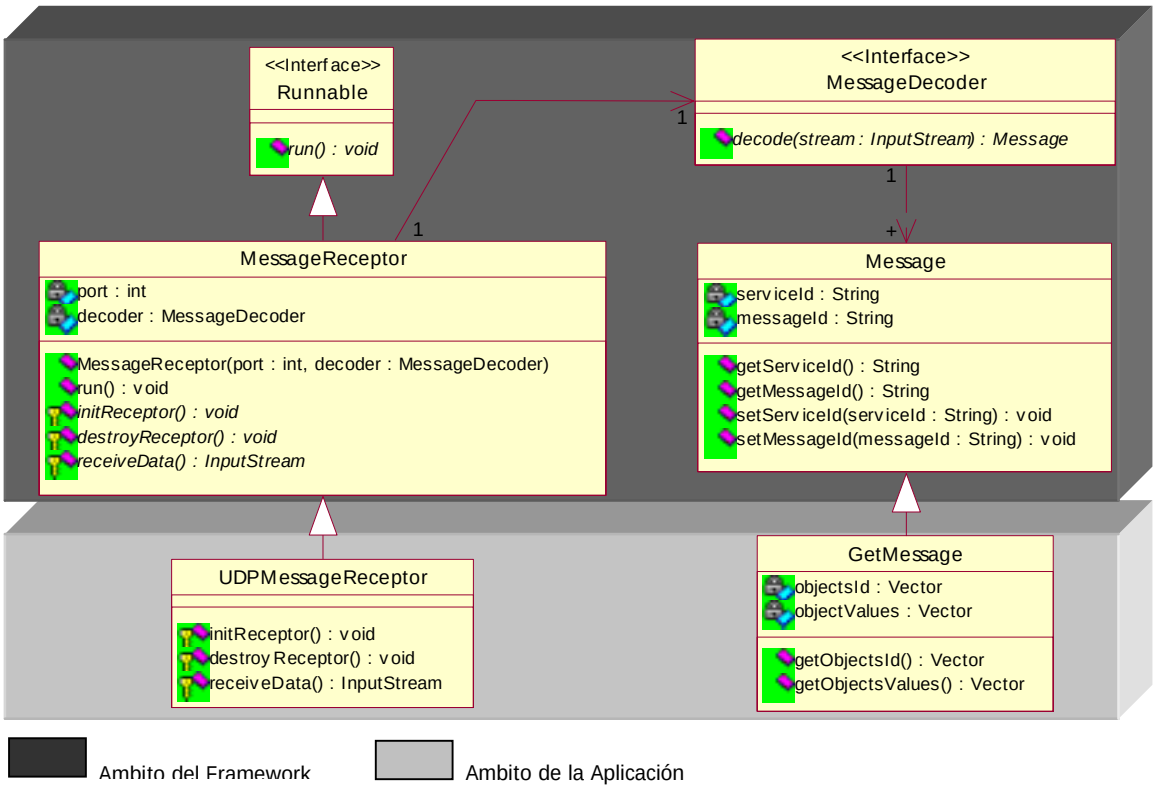


Figura 3.1 Ejemplo de un Framework Orientado a Objetos.

En este caso, las clases *MessageReceptor*, *MessageDecoder* y *Message* deben ser extendidas para crear una nueva aplicación, ya que estas tres clases tienen declarados métodos abstractos (Por ejemplo *initReceptor(…)*, *destroyReceptor(…)* y *receiveData(…)* de la clase *MessageReceptor*, y en general todos los métodos que aparecen en itálicas), lo cual representa la extensibilidad del framework. El método *run(…)* tiene la peculiaridad de ser el método principal de un hilo de control, y es ahí en donde se encuentra parte de la lógica de control del framework. La clase

3.3 Clasificación de los Frameworks

Los frameworks son clasificados ya sea de acuerdo a su ámbito de trabajo o a la técnica utilizada para extenderlos [Fayad-Schmidt, 1997]. La clasificación acorde al ámbito divide a los frameworks en frameworks de infraestructura de sistemas, frameworks de integración de middleware, y frameworks empresariales de aplicación. La clasificación acorde a las técnicas de extensión los divide en frameworks de caja blanca, frameworks de caja negra, y frameworks de caja gris.

3.3.1 Clasificación de los Frameworks de acuerdo a su Ambito

La clasificación de acuerdo a su ámbito divide a los frameworks en tres tipos:

- **Frameworks de infraestructura de sistemas**. Esta clasificación es simplemente la colección de frameworks portables y eficientes que soportan infraestructura de dominios, tales como los sistemas operativos, interfases de usuario, comunicaciones, etc.
- **Frameworks de middleware de integración**. Son diseñados para permitir al software modularizar, reutilizar y extender infraestructura de software para trabajar satisfactoriamente en un ambiente distribuido tales como ORBs (Object Request Brokers).
- **Frameworks de aplicaciones empresariales**. Dirigen un dominio de aplicación extenso tales como la banca, telecomunicaciones o medicina. Estos representan una piedra angular de aplicaciones de dominio

La Figura 3.2 nos muestra gráficamente esta clasificación.

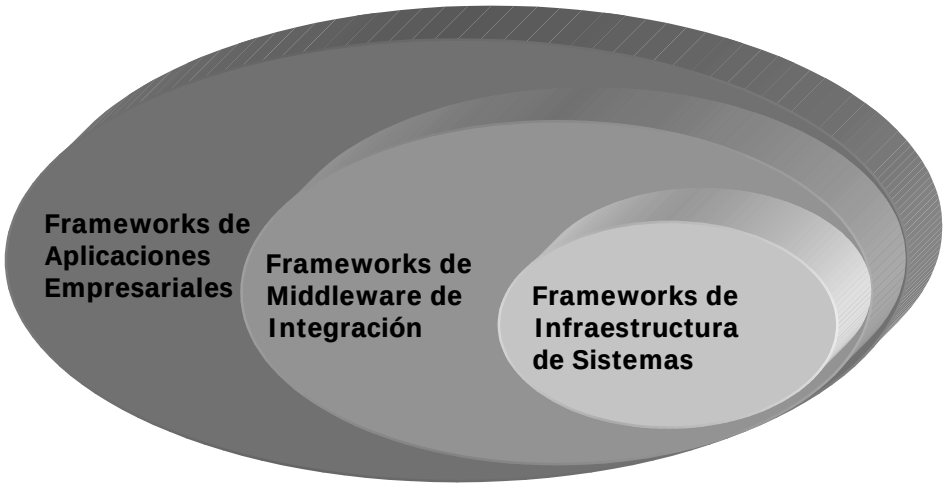


Figura 3.2 Clasificación de los Frameworks de acuerdo a su Ambito

En este caso, los Frameworks de Infraestructura de Sistemas dan soporte a los Frameworks de Middleware de Integración, ya que proporcionan la funcionalidad de bajo nivel como funciones de sistemas operativos, creación de conexiones de red, etc.

Los Frameworks de Middleware de Integración dan a su vez soporte a los Framework de Aplicaciones Empresariales, ya que proporcionan funcionalidades como protocolos de comunicación, bases de datos distribuidas, etc., quienes sirven para que múltiples aplicaciones de más alto nivel, como aplicaciones de contabilidad y uso de una red, aplicaciones que permiten transacciones bancarias en línea, etc., funcionen adecuadamente.

Dada esta clasificación, el framework que se pretende desarrollar corresponde a la clasificación de *Frameworks de Middleware de Integración*, debido a que permite implementar protocolos de administración de red para que los Agentes puedan comunicarse con los Administradores, así también proporciona funcionalidad para comunicar el estado de elementos de red. Cabe destacar que el framework a desarrollar no proporciona elementos de bajo nivel como establecimiento de conexiones de red, llamadas a funciones nativas de Sistemas Operativos, etc., por ende, no puede ser clasificado como un Framework de Infraestructura de Sistemas.

3.3.2 Clasificación de los Frameworks de acuerdo a su Técnica de Extensión

Existen tres categorías para lograr la extensibilidad del framework, estas son:

- **Frameworks de caja blanca**. En este tipo de framework, la extensibilidad es conseguida mediante la utilización de características orientadas a objetos tales como la herencia y el enlace dinámico. La funcionalidad existente es reutilizada y extendida ya sea heredando de las clases abstractas del framework o redefiniendo métodos predefinidos.
- **Frameworks de caja negra**. En un framework de caja negra la extensibilidad es lograda definiendo interfases para componentes que pueden ser "insertados" en el framework utilizando composición de objetos. La funcionalidad existente es extendida definiendo componentes que se adaptan a una interfaz en particular.
- **Frameworks de caja gris**. Un framework de caja gris es una mezcla entre el framework de caja blanca y el de caja negra. Los frameworks de caja gris permiten la extensibilidad utilizando herencia y enlace dinámico así como definiendo interfases.

La Figura 3.3 nos muestra gráficamente esta clasificación.

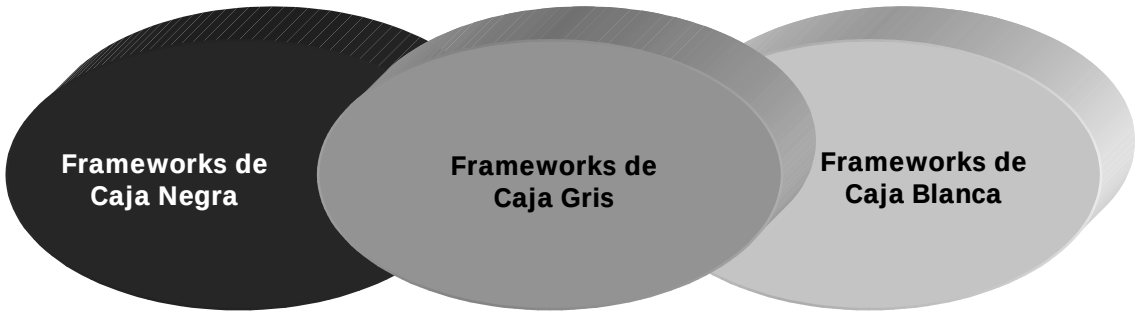


Figura 3.3 Clasificación de los Frameworks de acuerdo a su Técnica de Extensión

A continuación se presentan ejemplos de Frameworks de Caja Blanca, Caja Negra y Caja Gris.

3.3.2.1 Framework de Caja Blanca

La Figura 3.4 presenta un ejemplo de un framework de caja blanca.

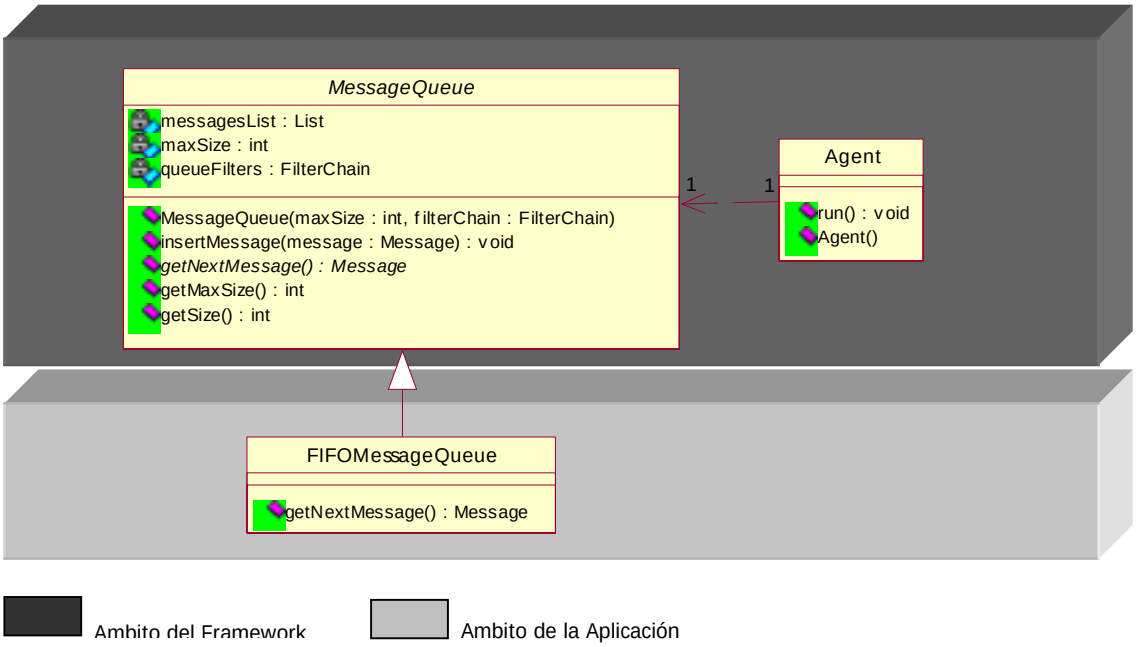


Figura 3.4 Ejemplo de un Framework de Caja Blanca.

El framework que es presentado en la Figura 3.4, es de caja blanca debido a que *MessageQueue* requiere ser extendida para proporcionar la funcionalidad de la aplicación. En este caso, el desarrollador de la aplicación requiere saber el tipo de funcionalidad esperado por la función abstracta *getNextMessage(…)*, para de ésta forma extender a la clase *MessageQueue* y formar la aplicación (En este ejemplo, el desarrollador de la aplicación crea la clase *FIFOMessageQueue*, la cual implementa a la función *getNextMessage(…)*).

3.3.2.2 Framework de Caja Negra

La Figura 3.4 presenta un ejemplo de un framework de caja negra.

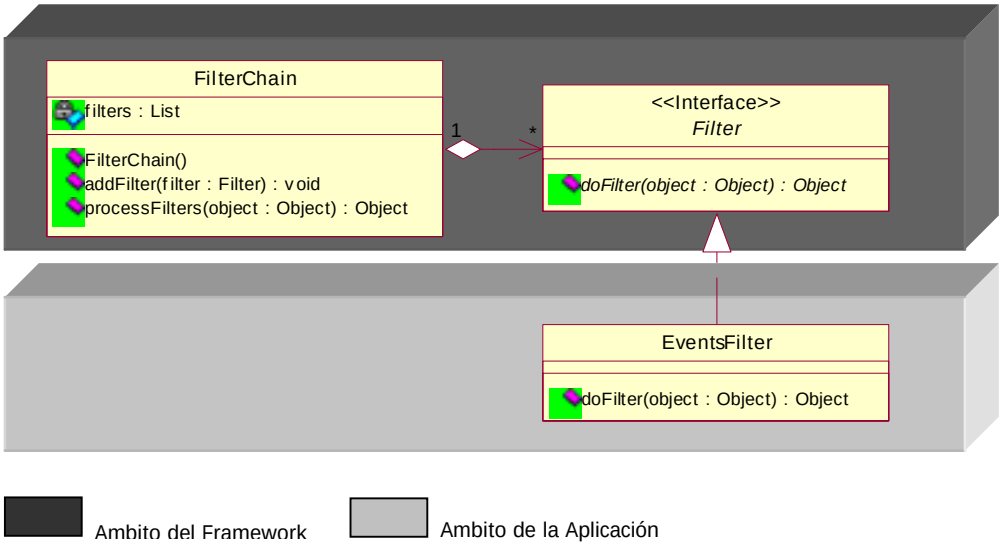


Figura 3.5 Ejemplo de un Framework de Caja Negra.

El framework que es presentado en la Figura 3.5, es de caja negra debido a que la interfaz *Filter* requiere ser implementada mas no extendida para proporcionar la funcionalidad de la aplicación. En este caso, el desarrollador de la aplicación no requiere saber de detalles de implementación, sino simplemente proporciona funcionalidad en el método *doFilter(…)*. En este ejemplo, el desarrollador de la aplicación crea la clase *EventsFilter*, la cual implementa a la función *doFilter(…)* de la interfaz *Filter*.

3.3.2.3 Framework de Caja Gris

La Figura 3.4 presenta un ejemplo de un framework de caja gris.

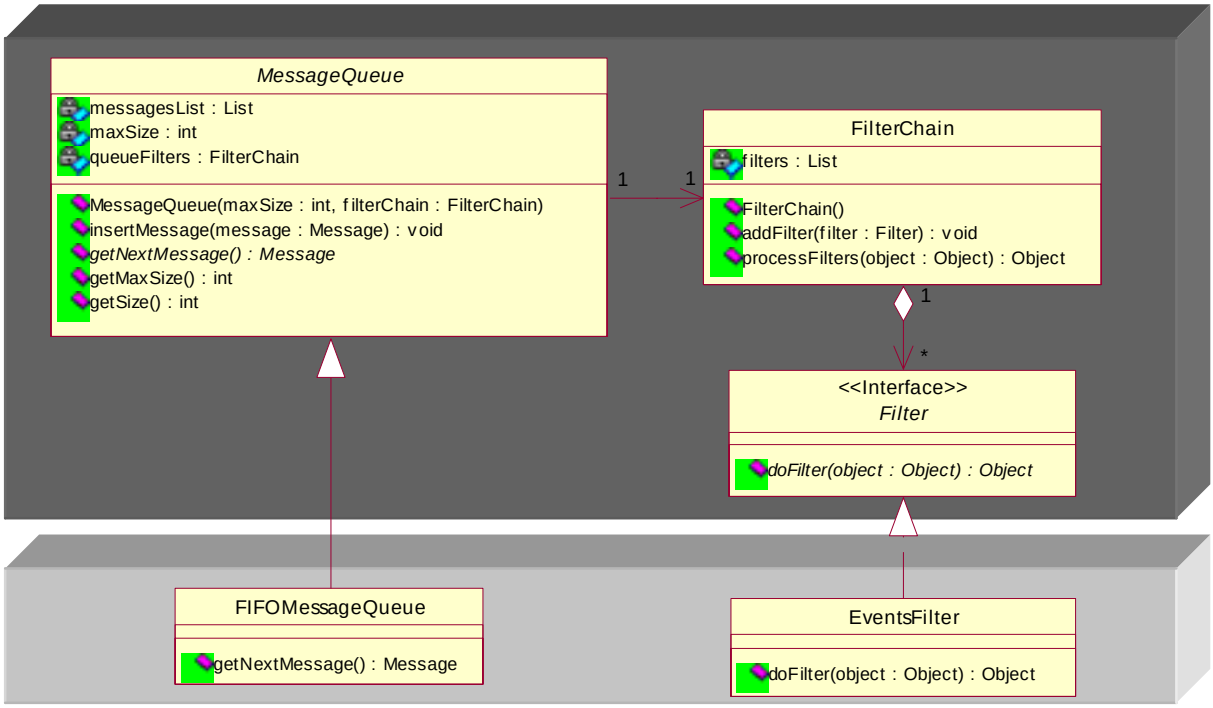


Figura 3.6 Ejemplo de un Framework de Caja Gris.

El framework que es presentado en la Figura 3.6, es la combinación de las técnicas utilizadas en los frameworks de caja blanca y caja negra ya que *MessageQueue* debe ser extendida para implementar la aplicación (En este caso se utiliza la herencia y por tanto se dice que es de caja blanca), mientras que la interfaz *Filter* requiere ser implementada mas no extendida para proporcionar la funcionalidad de la aplicación (En este caso se implementa la interfaz mas no se utiliza el mecanismo de herencia, por lo cual se dice que es de caja negra). En este caso, el desarrollador de la aplicación requiere saber el detalle y la funcionalidad esperada en la clase *MessageQueue* para implementar la subclase *FIFOMessageQueue*, mas no requiere saber de detalles de implementación de la interfaz *Filter*, sino simplemente proporciona funcionalidad en el método *doFilter(…)* de la clase *EventsFilter*.

3.4 Frameworks y Patrones de Diseño

Las abstracciones y los mecanismos de abstracción hacen del desarrollo de software más fácil, debido a que permiten al desarrollador trabajar con menos elementos. Es frecuentemente difícil reutilizar diseños de software o implementaciones directamente, porque las variaciones en los diferentes contextos de aplicación implican variaciones en los diseños e implementaciones. Cuando los diseños reusables e implementaciones son abstraídos de contextos específicos y se representan por conceptos y relaciones generalizadas, típicamente tienen un ámbito de aplicación mucho más amplio. A este tipo de abstracciones se les denomina abstracciones arquitecturales [Fayad, 1999].

Existen varias categorías de abstracciones arquitecturales, una de estas categorías se basa en el nivel de abstracción, en donde los patrones de diseño y los frameworks recaen. Los patrones de diseño son expresiones generales de temas recurrentes, los cuales han sido examinados y aplicados con anterioridad. Estos patrones de diseño se manifiestan como un conjunto de relaciones entre clases y un conjunto de comportamiento entre objetos que participan en el patrón.

Cada patrón de diseño sistemáticamente nombra, motiva y explica un diseño general que direcciona un problema de diseño recurrente en sistemas orientados a objetos. Describe el problema, la solución, cuando aplicar la solución y sus consecuencias. También brinda sugerencias y ejemplos de implementación. La solución en una conjugación de clases y objetos que resuelven el problema. La solución es adaptada e implementada para resolver un problema en un contexto o dominio en particular [Gamma, 1995].

Debido a que en un dominio específico, se necesitan resolver una serie de problemáticas, se utilizan un conjunto de patrones de diseño que son integrados en un framework que resuelve el dominio en específico. En conclusión, generalmente un framework utiliza patrones de diseño.

En la Figura 3.7 se muestra un ejemplo de un framework el cual abstrae la funcionalidad de la recepción de mensajes dado un protocolo de comunicación. En este framework, se tiene una entidad la cual está a la espera de recepción de datos, los cuales son convertidos en mensajes. En el framework, se utilizan dos patrones de diseño "*Template Method*" y "*Strategy*" [Gamma, 1995], los cuales se encuentran reflejados en el diagrama de clases de la Figura 3.7.

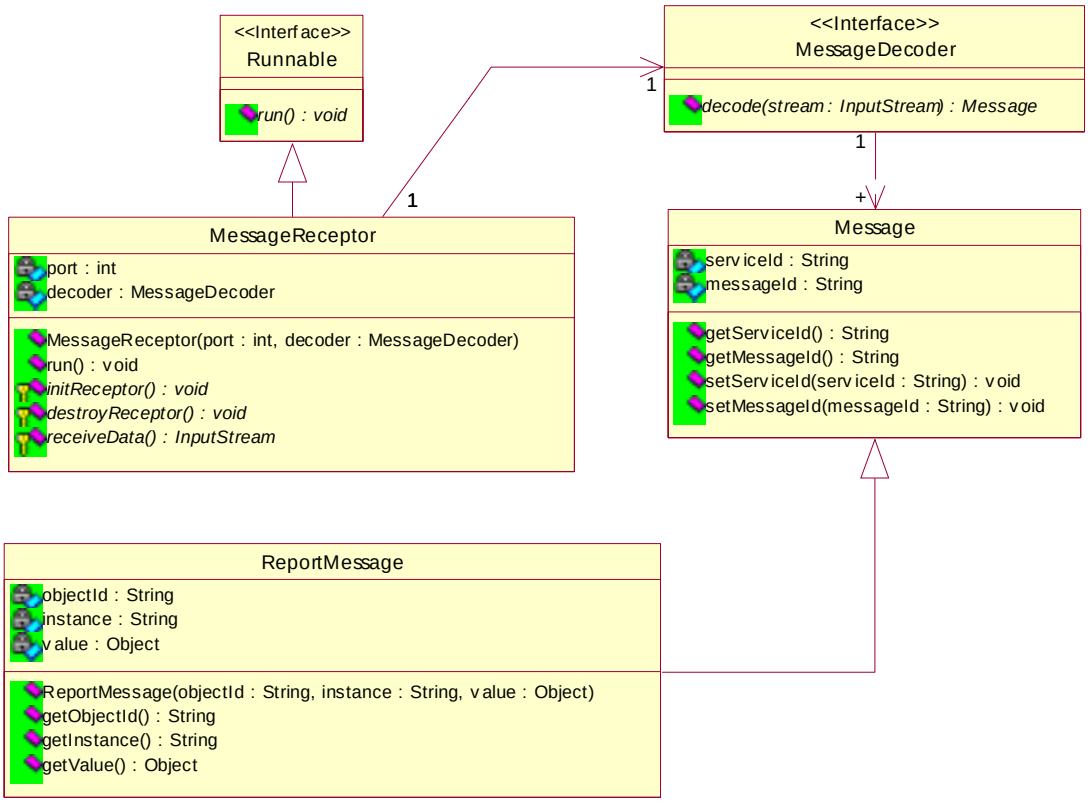


Figura 3.7 Ejemplo de un Framework que Aplica Patrones de Diseño

La aplicación de ambos patrones de diseño se describe a continuación:

- ***Template Method***. El método plantilla se encuentra en la clase "*MessageReceptor*" en su método *run()*, el cual contiene la lógica suficiente para iniciar y ejecutar el mecanismo de recepción de datos. Estos mecanismos deben ser implementados por el desarrollador del Agente a través de los métodos *initReceptor(…)*, *receiveData(…)*, y *destroyReceptor(…)*.

- ***Strategy***. El método plantilla delega a un objeto de la interfaz *MessageDecoder* la responsabilidad de convertir el flujo de información enviada por el Administrador a un Mensaje, el cual pueda ser interpretado por el Agente. En este caso el desarrollador del Agente debe implementar el método *decode(…)* en una clase concreta que implemente a ésta interfaz. Cabe destacar que instancias de la interfaz *MessageDecoder* deben tener la lógica suficiente de producir instancias de subclases de *Message* lo cual permitirá que existan varios formatos de mensajes.

3.5 Frameworks y su Utilización

Existen varias formas para utilizar un framework. Algunas maneras requieren un conocimiento profundo del framework que otras. Todas ellas son diferentes de la manera usual de desarrollo de software que utiliza la tecnología orientada a objetos, dado que todas ellas fuerzan a una aplicación a adaptarse al framework. Esto es, el diseño de la aplicación debe comenzar con el diseño del framework [Fayad, 1999].

Una aplicación desarrollada utilizando un framework tiene tres partes: el framework, las subclases concretas de las clases del framework, y clases auxiliares que apoyan ya sea a la funcionalidad del framework o a la funcionalidad de la aplicación. La aplicación desarrollada por el framework usualmente incluye un escrito que especifica que clases concretas serán utilizadas y como ellas serán interconectadas. También debería incluir objetos que no tienen relación con el framework o que usan uno o más objetos del framework. Los objetos que son llamados por objetos del framework tendrán que participar en el modelo colaborativo del framework y por lo tanto son parte del framework.

La manera más fácil para utilizar un framework es conectar componentes existentes. Esto no cambia el framework o hace nuevas subclases concretas. En este caso se reutilizan las interfaces del framework y las reglas para conectar los componentes. Esto es parecido a construir una tarjeta de circuitos al conectar circuitos integrados. Los programadores de aplicación sólo tienen que conocer que objetos del tipo X tienen que conectarse con objetos del tipo Y; ellos no tienen que conocer la especificación exacta de X y Y.

No todos los frameworks pueden trabajar de esta forma. Algunas veces cada nueva utilización del framework requiere de nuevas subclases. Esto conlleva a otra forma de utilizar un framework, la cual es definir nuevas subclases concretas las cuales se utilizan para implementar una aplicación. Las subclases son fuertemente acopladas a sus superclases, por tanto esta forma de utilizar un framework requiere mas conocimiento sobre las clases abstractas que la primera forma. Las subclases deben tener la especificación implicada por las superclases, por lo tanto el programador debe entender las interfaces del framework en detalle.

La forma de utilizar un framework que requiere el más amplio conocimiento es extenderlo para cambiar las clases abstractas que forman el "corazón" del framework, usualmente añadiendo nuevas operaciones o variables. Esta forma es como rellenar un esqueleto de una aplicación. Usualmente requiere el código fuente de un framework. A pesar de que es la manera más difícil de utilizar un framework, es también la más poderosa. Por otra parte, cambios a las clases abstractas puede romper con clases concretas existentes, y esta forma no funcionará cuando el principal propósito del framework es construir sistemas abiertos.

No obstante que en secciones anteriores se vio la clasificación de los frameworks de acuerdo a su forma de extensión, es preciso ver de una manera más clara esta clasificación. Si los programadores de aplicación pueden utilizar un framework conectando componentes

sin tener conocimiento de su implementación, el framework es un framework de caja negra. Los frameworks que utilizan el mecanismo de herencia usualmente requieren más conocimiento por parte de los desarrolladores y son llamados de caja blanca. Los frameworks de caja negra son fáciles de aprender a usar, pero los frameworks de caja blanca son mucho más extensibles.

Todas estas formas de utilizar un framework requiere el mapeo de la estructura del problema en la estructura del framework. Un framework fuerza a la aplicación a reutilizar su diseño. Los métodos orientados a objetos existentes usualmente comienzan con un análisis del modelo y derivan el diseño de él, pero esto no funciona con los frameworks. Es por ello que se necesitan nuevos métodos orientados a objetos que cubran esta necesidad.

3.6 Ingeniería de Dominio Orientada a Objetos

No obstante que hasta el momento no existe una metodología estándar para el desarrollo de frameworks, la Ingeniería de Dominio Orientada a Objetos (IDOO) es un método que permite generar frameworks de dominio específico [Chan 1998]. Las fases de este método son:

- **Análisis de Dominio**. Permite obtener los principales conceptos del dominio, los abstrae y determina las relaciones entre estos conceptos y determina los casos de uso del dominio. Además delimita el ámbito del dominio y las aplicaciones a las cuales el framework dará soporte.

- **Análisis de Variabilidad en los Componentes del Dominio**. Esta actividad no es parte de la IDOO, no obstante, la presente tesis la desarrolla para entender de manera clara las características comunes y de variabilidad de cada uno de los conceptos del dominio, lo cual marca el límite entre el corazón del framework (las similitudes) y las aplicaciones (la variabilidad), utilizando un modelo de características.

- **Diseño de Dominio**. Sobre la base del análisis de dominio y del análisis de variabilidad, se modela tanto el comportamiento estático como dinámico del framework, para esto se utilizan los patrones de diseño que se describen en [Gamma, 1995]. También se determinan las reglas de diseño que las aplicaciones deben de cumplir para utilizar el framework. Además se permite ver con mayor claridad la configuración que se debe de llevar a cabo en el framework en la adaptación de las aplicaciones.

- **Implementación del Framework de Dominio**. Dado que en esta etapa se tiene el diseño del framework de dominio, se escoge un lenguaje de programación orientado a objetos para codificar las clases presentadas en el diseño de dominio, lo cual termina con la etapa de desarrollo del framework, mas no con el desarrollo de las aplicaciones.

- **Aplicación del Framework de Dominio**. Una vez implementado el framework se presenta un ejemplo de la aplicación del framework, presentando su diseño y codificación.

Cada una de las etapas de la Ingeniería de Dominio Orientada a Objetos se describen con mayor detalle en los siguientes capítulos de la tesis. No obstante la Figura 3.8 muestra a grandes rasgos los entregables de cada una de éstas etapas.

Figura 3.8 Fases de la Ingeniería de Dominio Orientada a Objetos.

3.7 Conclusiones

En el siguiente capítulo se tienen las siguientes conclusiones:

- Se puede concluir que los frameworks de dominio orientados a objetos permiten abstraer un dominio mediante clases, relaciones y objetos, los cuales desempeñan la funcionalidad del dominio seleccionado.

- Los frameworks permiten cambiar el paradigma de desarrollo de aplicaciones, ya que en vez de que una aplicación tenga su propio control, el framework es el que tiene el control de la aplicación[8].

- No obstante que los frameworks permiten crear abstracciones poderosas para un dominio en específico, desarrollar un framework no es una tarea trivial, ya que se necesita un conocimiento profundo del dominio, patrones de diseño, análisis y diseño orientado a objetos para poder desarrollarlo.

[8] A lo cual se le denomina el principio de Hollywood: "no nos llames, nosotros te llamamos".

Capítulo

4

Análisis de Dominio Orientado a Objetos

4.1 Introducción

El Análisis de Dominio Orientado a Objetos (ADOO) es una parte fundamental en el desarrollo de Frameworks de Aplicación de Dominio Específico [Fayad 2, 1999]. Mediante el ADOO se podrán determinar las metas y objetivos del framework, su dominio, el dominio de la aplicación, los elementos que intervienen en el dominio del framework, sus conceptos, asociaciones y eventos. En este capítulo se presentan todos estos elementos, los cuales serán la base para poder desarrollar el framework.

Cabe señalar como un punto importante, que en esta etapa de desarrollo del framework, en ningún momento se modelan clases u objetos, ya que el principal objetivo del ADOO es el de encontrar y modelar los conceptos de un dominio dado, no los componentes de software que serán diseñados. Estos componentes de software serán diseñados en la etapa del Diseño de Dominio Orientado a Objetos (DDOO).

4.2 Definición

El análisis de dominio es el proceso de identificación y organización de conocimiento sobre algunas clases de problemas (lo cual es el dominio del problema) para soportar la descripción y solución de esos problemas [Prieto, 1991]. El proceso de resolución del problema que el análisis de dominio pretende, es utilizado para soportar estructuras conceptuales en cada dominio del problema.

Intuitivamente, el análisis de dominio es equivalente al análisis y especificación de requerimientos convencional en un metanivel con respecto a la construcción de software. El análisis de dominio trata con la identificación, adquisición, y representación de (potencialmente reusables) especificaciones e implementación del conocimiento por medio del software sobre la base de clases de problemas del mundo real.

En la programación orientada a objetos, la identificación de objetos, operaciones y sus relaciones en sistemas grandes no es una tarea trivial. Frecuentemente se tiene que lidiar con objetos a diferente nivel de abstracción, definir límites de abstracción, encontrar relaciones complejas, y encontrar todas las posibles operaciones desempeñadas en un objeto dado. Se tiene además, que identificar las clases de objetos e identificar sus atributos comunes. Todas estas tareas son pertinentes al análisis de dominio orientado a objetos (ADOO).

El ADOO crea un modelo genérico de objetos de dominio y casos de uso de dominio, los cuales abarcan múltiples aplicaciones del dominio así como también el dominio a través del tiempo [Fayad 2, 1999].

4.3 Descripción de la Fase de Análisis de Dominio Orientado a Objetos

En el proceso del ADOO, se identifican una serie de actividades que son llevadas a cabo, así como los entregables en cada una de ellas. No obstante que la presente tesis se basa en el método de ingeniería de dominio orientada a objetos (IDOO) [Chan, 1998], en esta fase se tomarán en cuenta las actividades presentadas en la Figura 4.1, la cual además de ilustrar el proceso del ADOO en términos de sus actividades, muestra sus entregables, su flujo de información y sus dependencias.

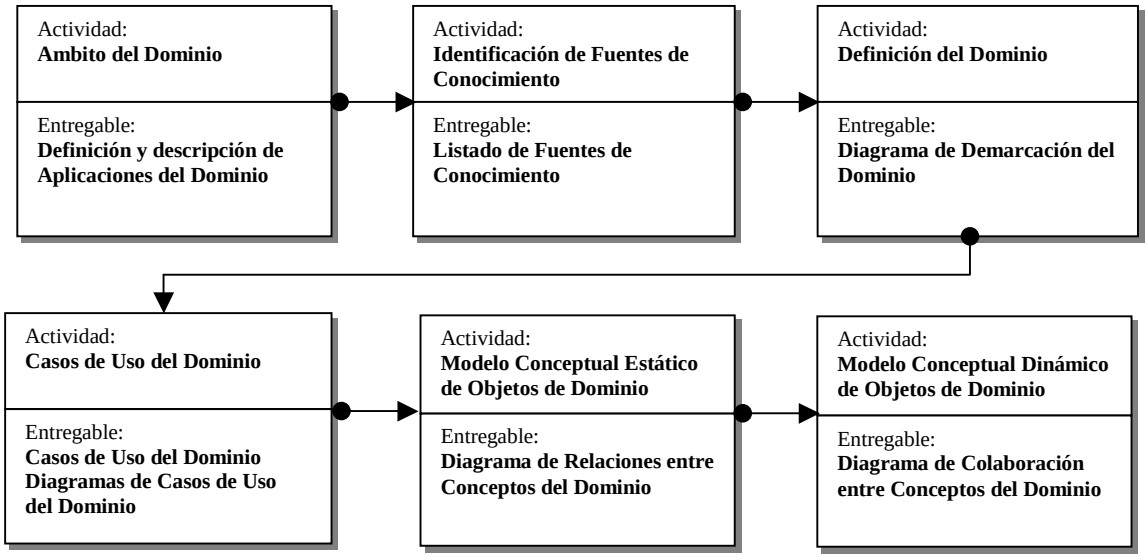


Figura 4.1 Proceso de Análisis de Dominio Orientado a Objetos.

Las actividades a desarrollar son las siguientes:

- **Ambito del Dominio**. La selección de un dominio apropiado es crítica para un proyecto de IDOO. La clave en la selección del ámbito del dominio reside en la definición de las aplicaciones que serán construidas en el dominio, lo cual permite definir el alcance, las metas y objetivos que el proyecto de IDOO tendrá que llevar a cabo.
  Entregable. El entregable en esta actividad es la definición y descripción de la serie de aplicaciones que el proyecto de IDOO permitirá desarrollar, así como las metas y objetivos pretendidos.

- **Identificación de Fuentes de Conocimiento**. El conocimiento sobre un dominio del problema y como implementar soluciones de software intensivas a problemas en el dominio pueden ser adquiridos de diferentes fuentes. Típicamente son las siguientes:
  - *Literatura técnica*: Libros, manuales, publicaciones científicas, etc.
  - *Aplicaciones existentes*: Códigos fuente de programas, documentación de diseño, manuales de usuario, etc.
  - *Análisis de mercado*.
  - *Experiencia humana en el dominio del problema:* La experiencia de un profesionista en el diseño de sistemas en el dominio (experiencia de analistas de sistemas, diseñadores, programadores, etc.).
  - *Registros históricos de la evolución en el dominio.*

  Entregable.
  El entregable en esta actividad es la definición y descripción de aquellas fuentes que serán utilizadas para la obtención de conocimiento del dominio.

- **Definición del Dominio**.
  La definición del dominio delimita el dominio en términos de su demarcación lógica, sus actores, sus eventos y las interfaces entre el y otros dominios.
  Entregable.
  El entregable en esta actividad es el diagrama de demarcación del dominio, en el cual se describen gráficamente tanto los actores, eventos e interfaces que participan en el dominio.

- **Casos de Uso del Dominio**.
  Los casos de uso del dominio son uno de los entregables fundamentales en la IDOO. Estos son descripciones abstractas de procesos de negocio que se aplican a un dominio en su totalidad en vez de a una aplicación en específico. Son desarrollados sobre la base de las fuentes de conocimiento elegidas para el desarrollo del proyecto de IDOO.
  Entregable.
  El entregable en esta actividad son tanto los casos de uso del dominio, así como los diagramas de casos de uso de domino.

- **Modelo Conceptual Estático de Objetos de Dominio**.
  El modelo conceptual estático de objetos de dominio muestra gráficamente las relaciones que tienen los conceptos que participan en el alcance del dominio. Este modelo es la base tanto para la identificación de patrones en el dominio como para la agregación de clases dentro de los subsistemas.
  Entregable.
  El entregable de esta actividad es la esquematización del modelo conceptual estático de objetos de dominio, el cual contendrá aquellos conceptos que pertenecen al dominio, sus relaciones, así como una base para la identificación de patrones en el dominio.

- **Modelo Conceptual Dinámico de Objetos de Dominio**.
  El modelo conceptual dinámico de objetos de dominio muestra eventos, colaboraciones, y relaciones de tiempo entre los conceptos sobre el curso de un caso de uso del dominio. Este modelo demuestra si todas las responsabilidades han sido bien asignadas y distribuidas entre los conceptos.
  Entregable.
  El entregable de esta actividad es la esquematización del modelo conceptual dinámico de objetos de dominio, el cual contendrá aquellos conceptos y eventos que intervienen en los diferentes casos de uso del dominio.

4.4 Ambito del Dominio

El establecimiento del ámbito del dominio es la primera actividad que se debe llevar a cabo al realizar el ADOO. En primer lugar se definen los tipos de aplicaciones a las cuales el framework dará soporte. Posteriormente se definen las metas, alcances y objetivos que en el desarrollo del framework se pretenden alcanzar.

4.4.1 Familia de Aplicaciones

La familia de aplicaciones que cubren el ámbito de dominio son:

- **Aplicaciones que asuman el rol de Agente en el contexto de la Administración de Redes**. Esta familia de aplicaciones estará constituida por Agentes de Administración de Red, los cuales interpreten y ejecuten comandos provenientes de un Administrador de Red. Los comandos estarán basados en los protocolos SNMP o CMIP (según sea el caso). Así también los Agentes reportarán alarmas en caso de que un Recurso bajo su cargo presente un estado excepcional.

4.4.2 Alcance, Metas y Objetivos

Alcance. El alcance del framework, estará delimitado con respecto al rol que desempeña un Agente, quien es responsable de funciones como la obtención, modificación, borrado y creación de los objetos administrados, teniendo además soporte para el reporte de alarmas que se presentan en los diferentes recursos del elemento de red que está siendo representado

por el Agente. Cabe destacar que no se diseñará ninguna funcionalidad que esté relacionada con el rol del Administrador.

Meta. Se tiene como meta primordial el generar un framework de aplicación orientado a objetos, el cual permita el desarrollo de las de aplicaciones descritas en la sección "Familias de Aplicaciones". Se deberá también

Objetivos. Se tiene como objetivo la generación de diseños que permitan la implementación de un framework orientado a objetos para el desarrollo de Agentes de Administración de Red basados en el modelo de información de administración de OSI y de Internet. También se pretende documentar de manera muy precisa cada una de las actividades y/o procesos realizados para el desarrollo del framework.

Para mostrar de una clara todas los puntos presentados en esta actividad, la Figura 4.2 esquematiza el ámbito del dominio del framework a desarrollar.

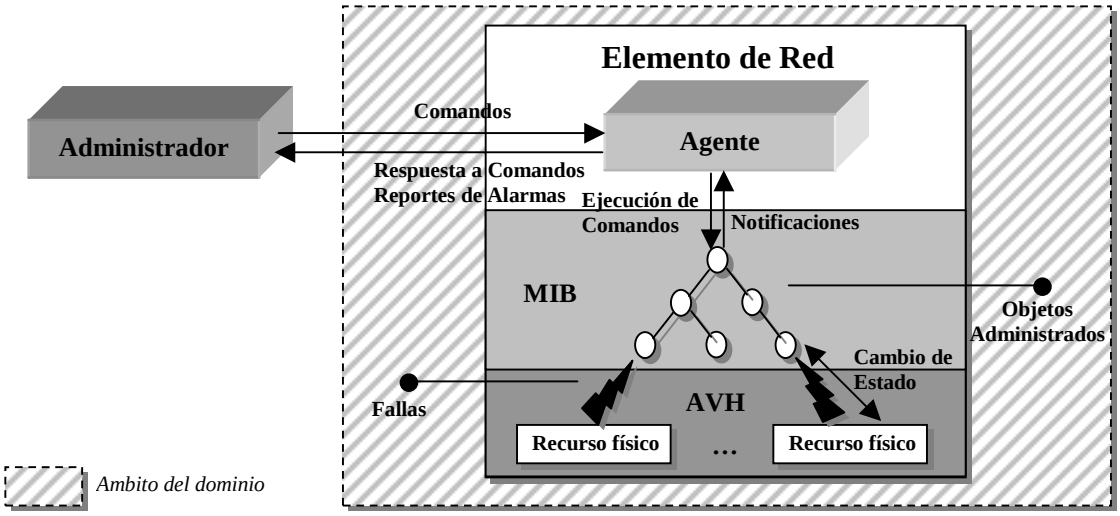


Figura 4.2 Ambito del Dominio.

4.5 Identificación de Fuentes de Conocimiento

Las fuentes de conocimiento del dominio con las cuales contará la tesis son principalmente literatura técnica, ya que otras fuentes como un experto en el dominio o código fuente de aplicaciones de administración de red son difíciles de encontrar o adquirir. Las fuentes de conocimiento identificadas son las siguientes:

Telecommunications Management Network……..……...[Udupa, 1999]
Este libro presenta un panorama general de la administración de redes de telecomunicaciones (TMN), varios conceptos y modelos que son utilizados en TMN, así como información relevante en las funciones desempeñadas por agentes, además de

algunos protocolos de administración de red que son utilizados para la comunicación entre los agentes y administradores. Presenta también un breve resumen de lo que es la supervisión de alarmas y sus funciones.

Network Management Standars……...[Black, 1992]
En este libro se describen de manera detallada los protocolos de administración de red más utilizados hasta la actualidad además de los funciones que soportan a cada uno de los comandos de los protocolos.

Sub-System Alarm Surveillance Ensemble……...[NMF 037, 1995]
Este manual técnico presenta detalladamente cada una de las funciones, eventos y actores que intervienen en la supervisión de alarmas. Plantea de manera breve aquellas clases de objetos administrados que tienen que ver en la supervisión de alarmas, y hace referencia a los manuales técnicos que las describen.

Recommendation X.721. **Definition of Management Information**.[CCITT X.721, 1992].
Este manual técnico define y describe detalladamente todas aquellas clases de objetos administrados que intervienen en cada una de las funciones de administración de red.

RFC 1157. **A Simple Network Management Protocol (SNMP)**……..[RFC 1157, 1990]
Este manual técnico describe la arquitectura empleada por uno de los protocolos de administración de red, SNMP. Presenta también las diferentes especificaciones del protocolo así como sus operaciones.

Domain-Specific Application Frameworks……..[Fayad 2, 1999]
Este libro presenta brevemente el diseño de un framework orientado a objetos dedicado a la implementación de interfases de administración de red en las redes de telecomunicaciones.

4.6 Definición del Dominio

Esta actividad brinda un panorama detallado del dominio. Dado que el dominio está delimitado por las actividades llevadas a cabo por los agentes, en la presente sección se deben tratar estos temas. No obstante, todas estas actividades se describieron en el capítulo "Agentes de Administración de Red", es por ello que en esta sección, sólo se construirá el diagrama de demarcación del dominio que se basa en dichas actividades descritas en ese capítulo.

4.6.1 Diagrama de Demarcación del Dominio

La Figura 4.3 muestra la demarcación del dominio que es pretendida por el framework. La demarcación del dominio está principalmente delimitada por tres capas funcionales. Estas capas se determinaron basándose en una inspección de cada uno de los procesos y conceptos del dominio que intervienen en él, en donde principalmente se observan tres clases de procesos, el proceso de monitoreo de fallas y cambio de estado en los Recursos

(Administración Virtual de Hardware), el control de los Objetos Administrados (MIB), y la interpretación y ejecución de comandos además del envío de reportes de fallas (Agente).

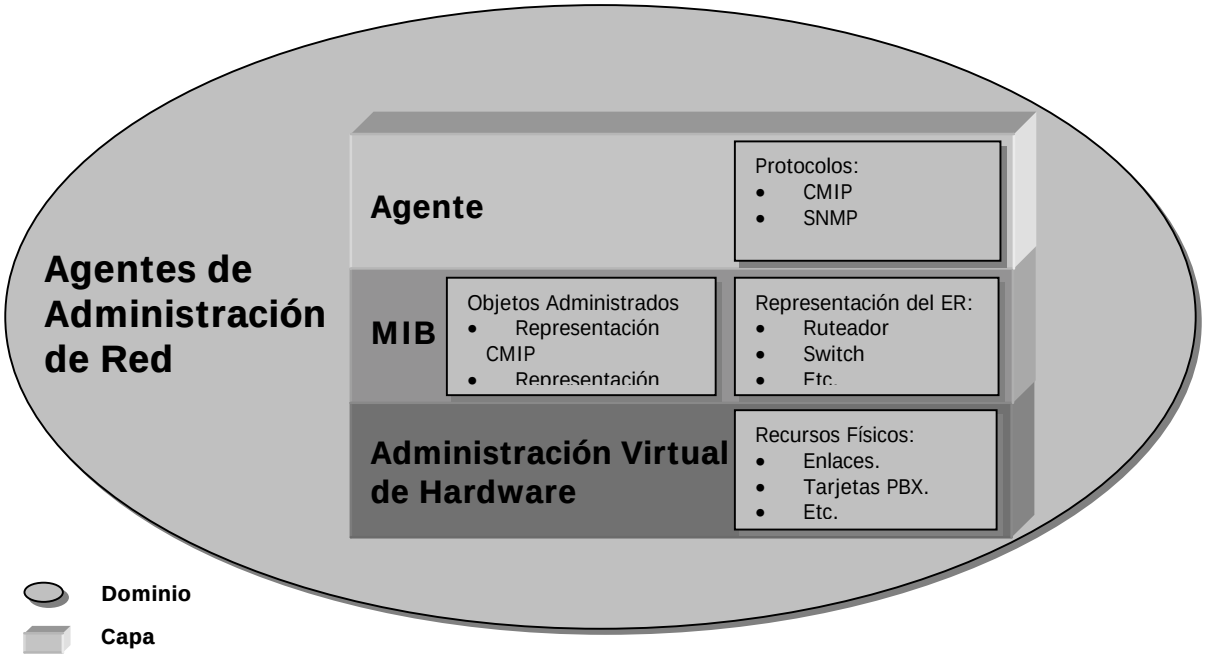


Figura 4.3 Diagrama de Demarcación del Dominio.

En el Diagrama de Demarcación se pueden observar para cada capa, las funcionalidades del dominio para los cuales el framework dará soporte, lo cual delimita o demarca el dominio.

En la capa Agente se dará soporte a los protocolos CMIP y SNMP, recibiendo, interpretando y ejecutando cada uno de los comandos que provengan de un Administrador, así también se enviarán comandos de respuesta. Además se reportan alarmas en caso de que se detecte una cambio de estado excepcional en los Objetos Administrados.

En la capa de la Base de Información de Administración (MIB), se dará soporte a los múltiples Objetos Administrados que pertenezcan al Elemento de Red (ER) representado por el Agente, es por ello que se podrán definir cualquier MIB que represente a un elemento de red, como ruteadores, multiplexores, switches, etc. Además dado que el Agente puede ser CMIP o SNMP, la representación de los Objetos Administrados estará sujeta al protocolo de Administración de Red que utilice el Agente.

Por último, en la Capa de Administración de Hardware se dará soporte a los diversos recursos físicos de un Elemento de Red, como lo pueden ser enlaces, conexiones, leds, etc., ya que ofrece una interfaz lógica a los recursos de hardware, transformando solicitudes de la capa MIB a solicitudes especificas en el hardware.

Además deberá existir un mecanismo tal que permita lanzar alarmas cuando el estado de un recurso físico sea excepcional, las cuales son reportadas a la MIB.

Una vez detectadas dichas capas en el dominio, intuitivamente se separan a su vez dos elementos, el elemento genérico y el específico. El elemento genérico es aquel que estará contenido dentro del framework, y que permitirá desarrollar el conjunto de aplicaciones deseado. El elemento específico estará implícitamente contenido en cada una de las aplicaciones que utilicen el framework. La Figura 4.4 nos muestra como es que las capas estarán definidas tanto en el ámbito del framework como en el de las aplicaciones.

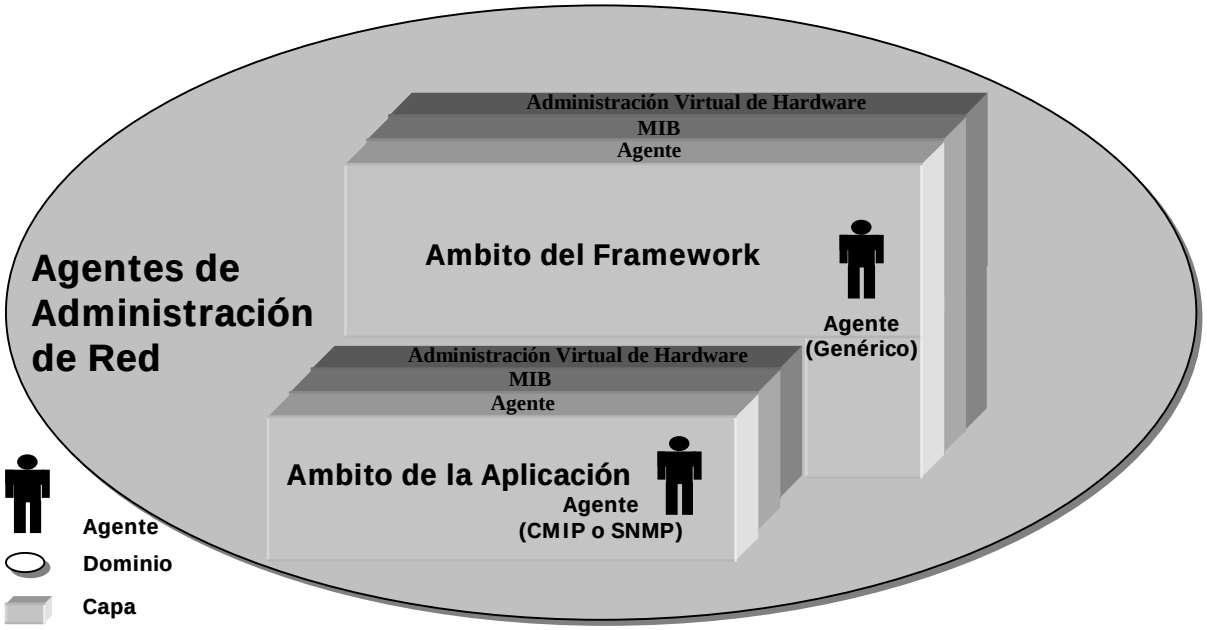


Figura 4.4 Ambito del Framework y de la Aplicación.

El ámbito de las aplicaciones que utilizan el framework, consiste entonces en definir cada una de las capas (Agente, MIB y AVH), las cuales forman un agente de administración de red en particular. Estas aplicaciones, extenderán las capas genéricas del framework y lo configurarán para permitir la funcionalidad deseada.

4.7 Casos de Uso

Una vez que delimitado el dominio, se pueden generar los casos de uso. Los casos de uso son descripciones de la secuencia de eventos que los actores generan en el sistema. Dado que en la presente tesis se debe diseñar un framework que permita utilizar tanto el modelo de OSI como el de Internet, se generaron los casos de uso pertenecientes a los dos modelos de información (Consúltese el "*Anexo A*" secciones "*A.2 Casos de Uso del Modelo OSI*" y "*A.3 Casos de Uso del Modelo de Internet*".). Los casos de uso se obtuvieron principalmente de las fuentes de conocimiento listadas con anterioridad.

4.8 Método de Abstracción de Casos de Uso

No obstante que al generar los casos de uso se pueden obtener los requerimientos de un sistema, en este caso se deben obtener los requerimientos de un framework, lo cual involucra a más de un sistema o subdominio (en este caso modelo de información de administración.). Es por ello que en la presente tesis se propone un método que permita obtener "Casos Abstractos de Uso", los cuales describan las acciones que se llevan a cabo en el dominio, independientemente del modelo de información. Cabe destacar que este método está basado parcialmente en la investigación realizada por [Ecklund, 1996], [Major, 1997], y [Miller, 1999] donde se proponen técnicas de abstracción de casos de uso, no obstante no orientan la abstracción de casos de uso entre subdominios similares, lo cual diferencia al presente método.

El método propuesto consta de dos pasos:

1. **Abstracción Vertical de los Casos de Uso**. Genera abstracciones de casos de uso que pertenecen a un mismo sistema o subdominio en particular. Sus principales objetivos son el analizar las similitudes entre los casos de uso, y abstraerlas de tal forma que se generen casos abstractos de uso, es decir, aquellos que sean similares entre sí, se agrupan y se vuelven uno solo. Se le denominó Abstracción Vertical debido a que solo toma en cuenta casos de uso que pertenecen a un mismo subdominio.

2. **Abstracción Horizontal de los Casos de Uso**. En este paso se agrupan los casos de uso que fueron resultado de la Abstracción Vertical y que tienen similitudes entre sí. Cada grupo debe contener a lo más uno y solo un caso de uso de cada subdominio[9]. El resultado son casos abstractos de uso que permiten observar todos los requerimientos del framework. Se le denominó Abstracción Horizontal debido a que toma en cuenta casos de uso pertenecientes a varios subdominios.

La Figura 4.5 muestra lo que el Método de Abstracción de Casos de Uso realiza.

[9] Dadas sus similitudes, si se pueden agrupar dos o más casos de uso de un mismo sistema o subdominio, entonces no se realizó correctamente la Abstracción Vertical.

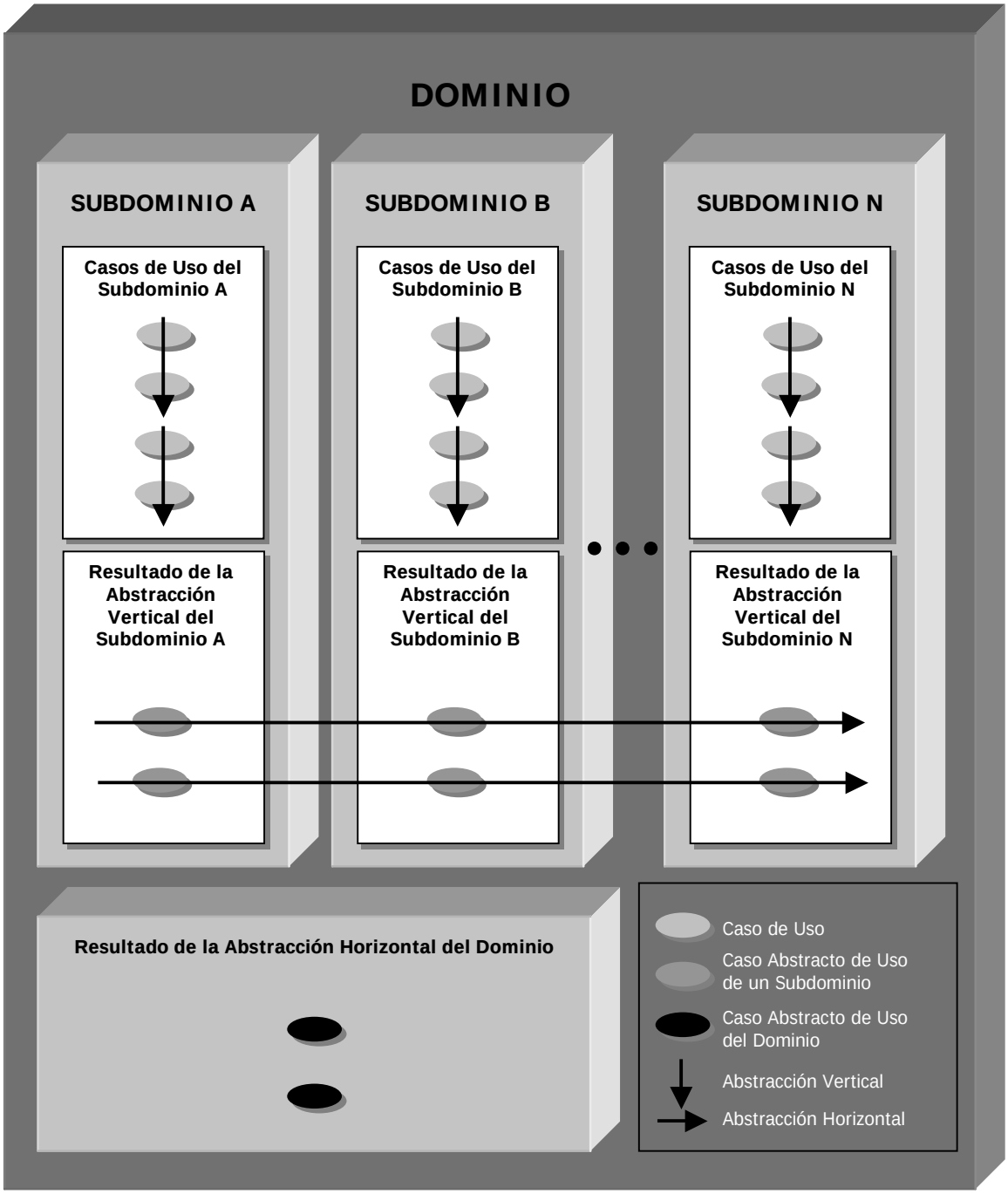


Figura 4.5 Método de Abstracción de Casos de Uso.

4.9 Abstracción Vertical de los Casos de Uso

En esta sección se explica con más detalle lo que se debe realizar en la Abstracción Vertical de los Casos de Uso. Esta se divide en cuatro pasos:

1. **Determinar los conceptos del dominio**. Los conceptos del dominio pueden ser componentes, agentes u objetos con los cuales existe una interacción u operación sobre ellos o que simplemente generan o ejecutan una acción en particular. En este paso pueden utilizarse tanto el "*Método del Listado de Categorías de Conceptos*" [Larman, 1999] ó el "*Método de Frases Nominales*"[Abbot, 1983].

2. **Determinar las acciones llevadas a cabo por cada caso de uso y su orden de ejecución**. Al igual que los conceptos del dominio, las acciones que se llevan a cabo en los casos de uso, se encuentran inmersas en la descripción de los mismos y tienen una interacción directa entre los conceptos. Las acciones las ejecutan tanto los Actores como el Sistema. El orden en que éstas se ejecutan, tiene que ver con la secuencia de pasos de cada caso de uso.

3. **Agrupar los casos de uso que tengan un comportamiento "similar"**. Una vez que se realizó el paso anterior, se agrupan los casos de uso que mantengan un orden similar en la ejecución de sus acciones, es decir, cada grupo tiene un patrón de comportamiento semejante.

4. **Generar casos abstractos de uso**. Los casos de uso que pertenecen a un mismo grupo se fusionan entre sí, lo cual da como resultado un caso abstracto de uso. Esto se realiza abstrayendo de manera secuencial y paralela las acciones de todos casos de uso que pertenecen a un grupo.

Cabe destacar que la Abstracción Vertical de los Casos de Uso se debe llevar a cabo para cada uno de los subdominios. En la presente tesis se llevará a cabo la Abstracción Vertical para cada modelo de información.

4.9.1 Abstracción Vertical de los Casos de Uso del Modelo OSI

El primer paso es el encontrar los conceptos del dominio, en la presente tesis se utilizó una combinación del "*Método del Listado de Categorías de Conceptos*" [Larman, 1999] y del "*Método de Frases Nominales*" [Abbot, 1983] para obtenerlos. La Tabla 4.1 presenta una relación de categorías y conceptos encontrados en las descripciones de los casos de uso del Modelo OSI.

Categorías	Conceptos
Sistemas Externos	Administrador
Sistemas Internos	Agente MIB Administración Virtual de Hardware
Administradores de Sistemas	Administrador Agente Administrador de la MIB Administrador Virtual de Hardware Administrador de Procesos
Contenedores	MIB Bitácora Contenedor de Recursos
Cosas dentro de un contenedor	Objeto Administrado Recurso Proceso
Eventos	Falla Cambio de Estado Solicitud
Reporte de Eventos	Alarma Mensaje Notificación Reporte de Resultado de una Acción Severidad de Alarma
Analizadores de Reportes	Administrador Discriminador de Envío de Eventos Analizador de Severidad de Alarma Analizador de Solicitudes Analizador de Permisos en Objetos Administrados Filtro de Procesamiento de Bitácora
Buscadores	Buscador de Objetos Administrados Buscador de Recursos
Colecciones de cosas	Lista de Identificadores de Solicitud Lista de Mapeo de Recursos Lista de Espera de Solicitudes

Tabla 4.1 Lista de Categorías y Conceptos del Modelo OSI.

Después de haber encontrado los conceptos del Modelo OSI, ahora se deben determinar las acciones ejecutadas por los conceptos. Esto se debe realizar para cada caso de uso. Para esto, se utilizó una especie de “Diagrama de Colaboración” solo que en vez de objetos se tienen conceptos y en vez de mensajes se tienen acciones. Se utilizó esta clase de diagramas debido a que mediante éstos es relativamente fácil identificar la secuencia y/o el orden en que se ejecutan las acciones en cada caso de uso, lo que permite a su vez identificar a grupos de casos de uso que pueden ser abstraídos. Los diagramas de colaboración del modelo OSI pueden ser consultados en el "*Anexo B*" en su sección "*B.2 Diagramas Conceptuales de Colaboración del Modelo OSI*”.

Como puede observarse, a excepción del diagrama de colaboración del caso de uso "*Reportar Eventos generados en los Recursos (Modelo OSI)*", todos los diagramas restantes tienen un patrón similar tanto en la aparición de sus conceptos como en la ejecución de sus acciones. Esto implica que este grupo de casos de uso se puede abstraer. Para realizar esto, se necesita abstraer de manera secuencial y paralela las acciones de todos los casos de uso que pertenecen a este grupo, es decir, cada acción que sea similar respecto a otras que, se deben abstraer, siempre y cuando pertenezcan a la misma secuencia. La Figura 4.6 muestra de manera gráfica la abstracción secuencial de las acciones.

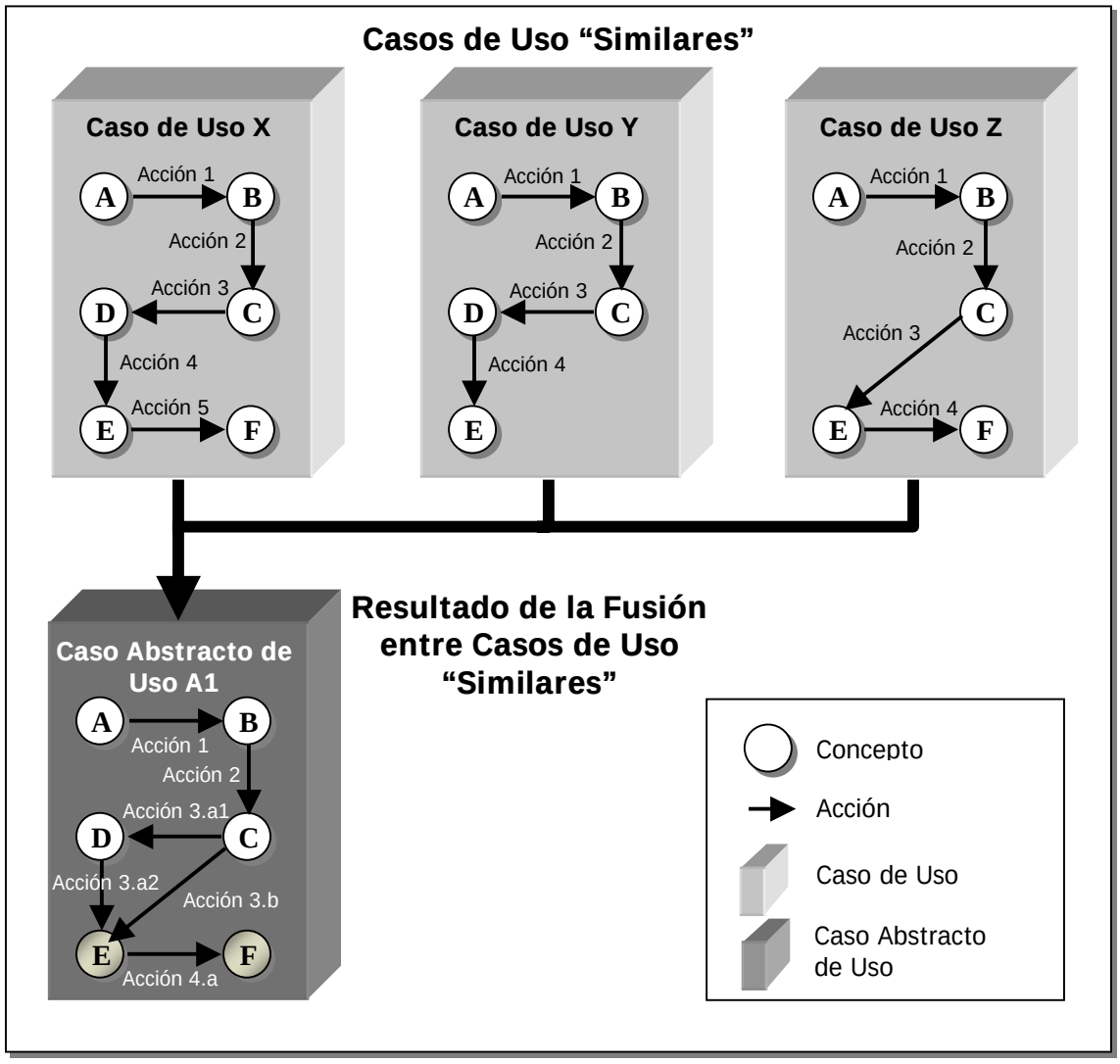


Figura 4.6 Abstracción Secuencial de Acciones en Casos de Uso "Similares".

Al realizar la abstracción secuencial de las acciones, se deben de tomar en cuenta los siguientes puntos:

1. En caso de que las acciones a realizar solo difieran por los parámetros de acción, estos se omitirán en la redacción de la acción, se omitirán también las referencias hacia dichos parámetros en acciones posteriores.
2. En caso de que en la secuencia de acciones de un caso de uso se omita la interacción con un concepto, el cual interviene en secuencias de otros casos de uso similares, se generará una acción general, la cual tendrá una bifurcación donde se especifique los casos en los cuales se omitirá interactuar con dicho concepto y cuando se tendrá que interactuar con el mismo.

El resultado de aplicar la Abstracción Vertical en el modelo OSI está reflejado en los casos abstractos de uso "*Ejecutar un Servicio en el Agente (Servicio CMIS)*" y "*Reportar eventos generados en los Recursos (Modelo OSI)*" (Véase el "*Anexo A*" en su sección "*A.4 Casos Abstractos de Uso del Modelo OSI*").

4.9.2 Abstracción Vertical de los Casos de Uso del Modelo de Internet

Una vez aplicada la Abstracción Vertical de los Casos de Uso del Modelo OSI, ahora es necesario realizar lo mismo con el modelo de Internet, por lo cual se deben encontrar los conceptos de este modelo de información. La Tabla 4.2 presenta una relación de categorías y conceptos encontrados en las descripciones de los casos de uso del modelo de Internet.

Categorías	Conceptos
Sistemas Externos	Administrador
Sistemas Internos	Agente MIB Administración Virtual de Hardware
Administradores de Sistemas	Administrador Agente Administrador de la MIB Administrador Virtual de Hardware
Contenedores	MIB Contenedor de Recursos
Cosas dentro de un contenedor	Objeto Administrado Recurso
Eventos	Falla Solicitud
Reporte de Eventos	Alarma Mensaje Notificación
Analizadores de Reportes	Administrador Analizador de Solicitudes Analizador de Permisos en Objetos Administrados
Buscadores	Buscador de Objetos Administrados Buscador de Recursos
Colecciones de cosas	Lista de Identificadores de Solicitud Lista de Mapeo de Recursos Lista de Espera de Solicitudes

Tabla 4.2 Lista de Categorías y Conceptos del Modelo de Internet.

Después especificar los conceptos del Modelo de Internet, ahora se determina el orden en que se ejecutan las acciones en cada caso de uso, para esto, consúltese el "*Anexo B*" en su sección "*B.3 Diagramas Conceptuales de Colaboración del Modelo de Internet*".

Como puede observarse, a excepción del diagrama de colaboración del caso de uso "*Reportar Eventos generados en los Recursos (Modelo de Internet)*", todos los diagramas restantes tienen un patrón similar tanto en la aparición de sus conceptos como en la ejecución de sus acciones. Esto implica que este grupo de casos de uso se puede abstraer, siguiendo el Método de Abstracción Vertical de Casos de Uso.

El resultado de aplicar la Abstracción Vertical en el modelo de Internet está reflejado en los casos abstractos de uso "*Ejecutar un Servicio en el Agente (Modelo de Internet)*" y "*Reportar eventos generados en los Recursos (Modelo de Internet)*" (Véase el "*Anexo A*" en su sección "*A.5 Casos Abstractos de Uso del Modelo de Internet*").

4.10 Abstracción Horizontal de los Casos de Uso

La Abstracción Horizontal, prácticamente realiza los mismos pasos que la Abstracción Vertical, la diferencia radica en que toma como entrada los Casos Abstractos de Uso que se generaron en la Abstracción Vertical. La Figura 4.7 muestra las entradas de ambas abstracciones, así como sus entregables.

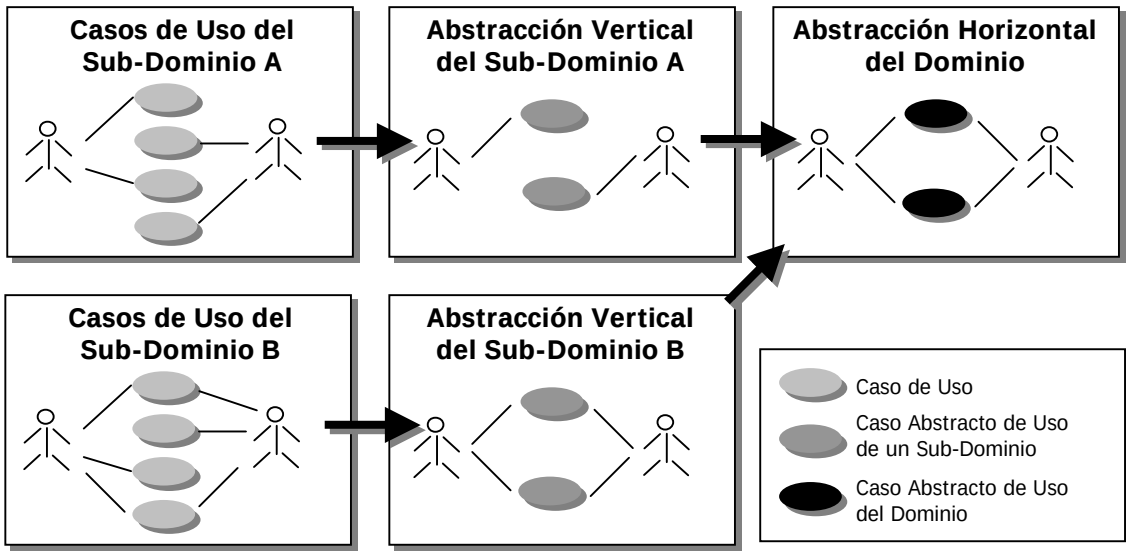


Figura 4.7 Abstracción Horizontal de los Casos de Uso.

A continuación se explica con más detalle lo que se debe realizar en la Abstracción Horizontal de los Casos de Uso. Esta se divide en cuatro pasos:

1. **Determinación de las acciones llevadas a cabo por cada caso abstracto de uso y su orden de ejecución**. Las acciones que se llevan a cabo en los casos abstractos de uso, se encuentran inmersas en la descripción de los mismos. Las acciones las ejecutan tanto los Actores como el Sistema. El orden en que éstas se ejecutan, tiene que ver con la secuencia de pasos de cada caso abstracto de uso.

2. **Agrupación de casos abstractos de uso que tengan un comportamiento "similar"**. Una vez que se realizó el paso anterior, se agrupan los casos de uso que mantengan un orden similar en la ejecución de sus acciones, es decir, cada grupo tiene un patrón de comportamiento semejante.

3. **Generación de casos abstractos de uso del dominio**. Los casos abstractos de uso que pertenecen a un mismo grupo se fusionan entre sí, lo cual da como resultado un caso abstracto del dominio. Esto se realiza abstrayendo de manera secuencial y paralela las acciones de todos casos abstractos de uso que pertenecen a un grupo.

4. **Esquematización de los Casos de Uso del Dominio**. Una vez que se obtuvieron los casos de uso del dominio, es necesario generar un diagrama en el cual los actores del sistema y los casos de uso del dominio sean claramente identificados, además de poder observar sus relaciones.

La Abstracción Horizontal da como resultado los requerimientos que el framework debe cumplir. En el dominio que se ha definido, se ha realizado la Abstracción Vertical tanto en el modelo de OSI como el de Internet.

4.10.1 Acciones de Casos Abstractos de Uso del Modelo OSI e Internet

En este punto, se toman como entrada los Casos Abstractos de Uso que se generaron en la Abstracción Vertical. Al igual que en un punto análogo de la Abstracción Vertical, se generan "*Diagramas de Colaboración*" para representar las interacciones existentes entre los conceptos (Véase el "*Anexo B*" en sus secciones "*B.4 Diagramas Conceptuales Abstractos del Modelo OSI*" y "*B.5 Diagramas Conceptuales Abstractos del Modelo de Internet*".).

4.10.2 Agrupación de los Casos Abstractos de Uso

Este punto permite agrupar casos abstractos de uso de cada uno de los subdominios (En este caso el modelo OSI y el modelo de Internet) para generar "*Casos Abstractos de Uso del Dominio*", los cuales describen los requerimientos del framework.

Al observar los diagramas de colaboración del modelo OSI y del modelo de Internet, existen similitudes entre dos pares de éstos. Estas similitudes se mencionan a continuación:

Similitud entre los casos abstractos de uso "*Ejecutar un Servicio en el Agente (Servicio CMIS)*" y "*Ejecutar un Servicio en el Agente (Servicio SNMP)*". Estos casos abstractos de uso permiten ejecutar acciones en el Agente, cuando un Administrador envía una solicitud al mismo. Otra similitud es que las acciones siempre se llevan a cabo en los Objetos Administrados, teniendo una interacción similar entre los conceptos abstractos del dominio.

Similitud entre los casos abstractos de uso "*Reportar eventos generados en los Recursos (Modelo OSI)*" y "*Reportar eventos generados en los Recursos (Modelo OSI)*". Estos casos abstractos de uso permiten reportar eventos ocurridos en Recursos administrados por el Agente hacia el Administrador. No obstante que algunos conceptos del modelo OSI no existen en el modelo de Internet, se pueden generar abstracciones que permitan incluir o excluir interacción con estos conceptos en los casos abstractos de uso del dominio.

4.10.3 Casos Abstractos de Uso del Dominio

El resultado de aplicar la Abstracción Horizontal en los modelos OSI e Internet está reflejado en los casos abstractos de uso "*Ejecutar un Servicio en el Agente* " y "*Reportar eventos generados en los Recursos*" (Véase el "*Anexo A*" en su sección "*A.6 Casos Abstractos de Uso del Dominio*").

Estos casos de uso representan los requerimientos formales del framework, los cuales deberán tomarse como base para el diseño del mismo.

4.10.4 Esquematización de los Casos de Uso del Dominio

Dado que se han generado los casos de uso del dominio, es necesario plasmar las relaciones entre los casos de uso y los actores del sistema. Para ello se presenta en la Figura 4.8a y Figura 4.8b el Diagrama de Casos de Uso del Dominio.

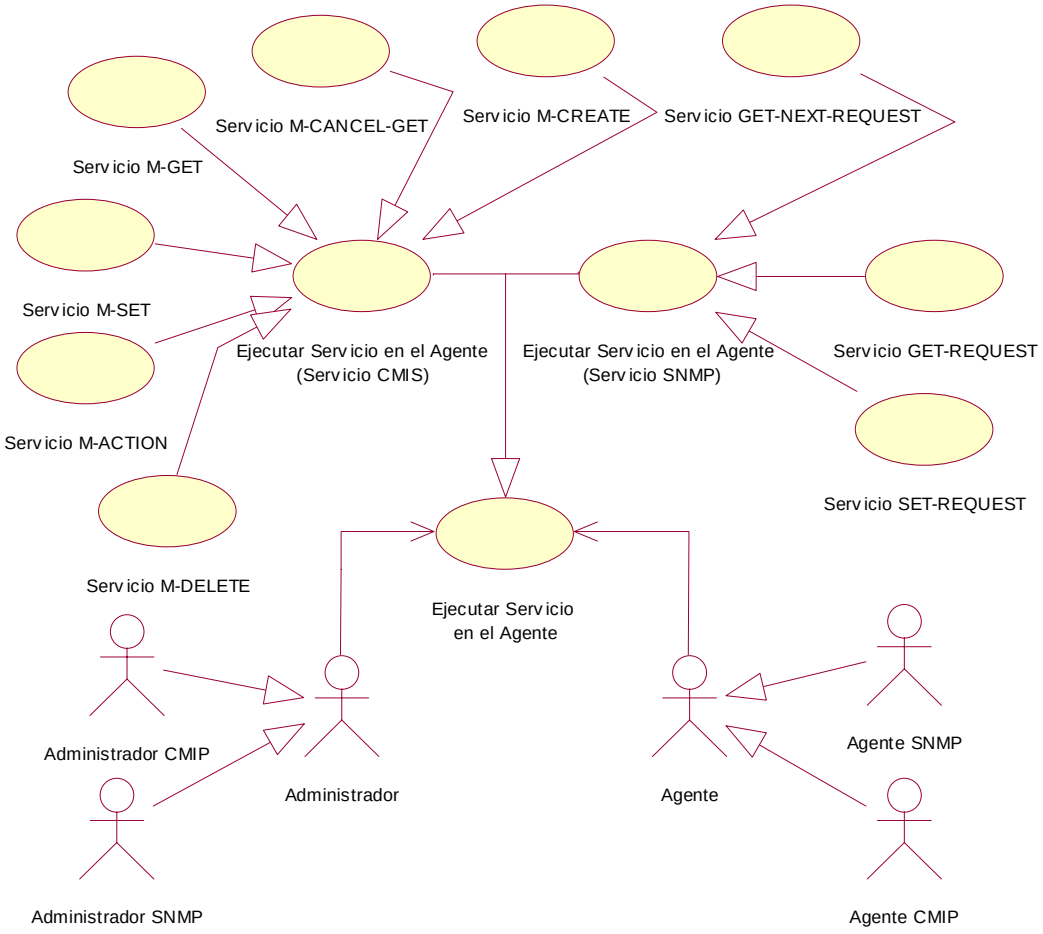


Figura 4.8a Diagrama de Casos de Uso del Dominio.

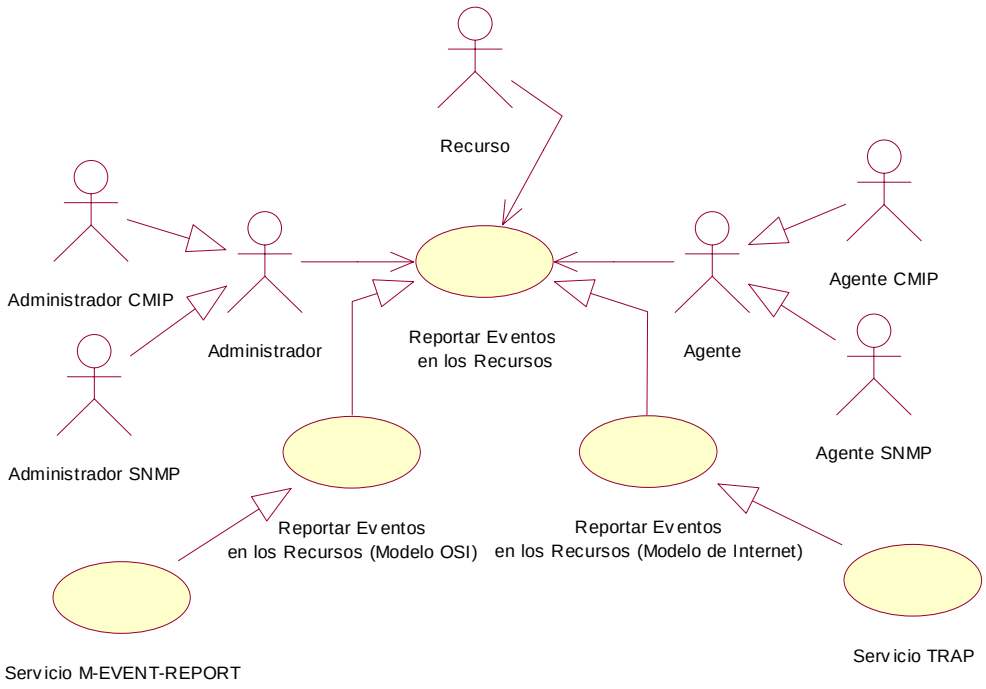


Figura 4.8b Diagrama de Casos de Uso del Dominio.

Las figuras anteriores permiten observar las abstracciones realizadas tanto en la "*Abstracción Vertical*" como en la "*Abstracción Horizontal*" de los casos de uso. Las abstracciones se reflejan principalmente en las relaciones de herencia entre los elementos que participan en el diagrama.

4.11 Modelo Conceptual Estático de Objetos de Dominio

Un entregable importante en proceso del ADOO es el Modelo Conceptual Estático de Objetos de Dominio (MCEOD). Este modelo permite identificar aquellos conceptos que intervienen en el dominio, sus atributos y las asociaciones entre éstos.

Cabe destacar que el MCEOD, trata con conceptos, no con clases, las cuales se derivarán de los conceptos en algún punto del Diseño de Dominio Orientado a Objetos (DDOO), es decir, el MCEOD es una descripción del dominio de un problema real, no es una descripción del diseño de software, aún cuando dichos conceptos sean componentes de software [Larman, 1999]. La creación del MCEOD consta tres pasos, los cuales son:

- Abstracción Conceptual del Dominio.
- Determinación de las Asociaciones entre los Conceptos.
- Esquematización del MCEOD.

Cada una de estas etapas se describen y aplican a continuación.

4.11.1 Abstracción Conceptual del Dominio

El primer paso que se debe realizar para poder crear el Modelo Conceptual Estático de Objetos de Dominio, es el realizar la Abstracción Conceptual del Dominio. Mediante ésta, es posible determinar conceptos genéricos los cuales pueden ser utilizados, para describir el dominio independientemente del modelo de información utilizado.

Para realizar la Abstracción Conceptual del Dominio, es necesario agrupar conceptos que tengan características similares. Para esto, se debe generar una tabla, la contenga en cada una de sus filas a conceptos similares entre los subdominios, las principales diferencias entre los conceptos, el concepto genérico derivado de ambos, y una explicación de la abstracción del concepto. Los conceptos son tomados de la fase "*Abstracción Vertical*" aplicada en cada modelo de información. Para determinar si un concepto es similar a otro, se deben observar las acciones que llevan a cabo, las interacciones con otros conceptos y la funcionalidad que pretenden realizar al ejecutar determinadas acciones.

La Tabla 4.3 muestra el resultado de realizar la Abstracción Conceptual del Dominio. Como puede observarse, existen algunos conceptos que son utilizados en el modelo OSI y que no existen en el modelo de Internet. No obstante, éstos deben existir, ya que el framework debe tomarlos en cuenta cuando se desarrolle una aplicación del modelo OSI.

Conceptos		Diferencias entre Conceptos	Concepto Genérico	Descripción del Concepto Genérico
Modelo OSI	Modelo de Internet			
Administrador	*Administrador*	Existe una gran diferencia entre la funcionalidad que brinda el Administrador del modelo OSI respecto al Administrador del modelo de Internet, no obstante dado que en la presente tesis el Administrador está fuera del ámbito de la misma, no se describirá la diferencia entre ambos conceptos.	*Administrador*	Encargado de realizar y encapsular solicitudes al Agente, a través de un protocolo de administración de Red.
Administrador de Procesos	*Administrador de Procesos*	Esta entidad no tiene diferencia entre los modelos ya que en los dos se necesita administrar procesos.	*Administrador de Procesos*	Entidad que permite generar y cancelar procesos.
Administración Virtual de Hardware	*Administración Virtual de Hardware*	En este caso no existe una diferencia entre ambos conceptos, ya que la Administración Virtual de Hardware es independiente del modelo de información.	*Administración Virtual de Hardware*	Encargada de la administración y monitoreo de los recursos de un elemento de red.
Agente	*Agente*	La principal diferencia entre estos conceptos es que el Agente de OSI utiliza el protocolo CMIP y el Agente de Internet utiliza el protocolo SNMP.	*Agente*	Encargado de recibir y dar seguimiento a las solicitudes del Administrador, independientemente de que protocolo de administración de red se esté utilizando.
Analizador de Permisos	*Analizador de Permisos*	En este caso, las diferencias entre ambos conceptos radican tanto en las entidades sobre las cuales se analizan los permisos (Objetos Administrados), y los tipos de permisos que se pueden otorgar sobre ellos.	*Analizador de Permisos*	Entidad que analiza los tipos de permisos que un Objeto Administrado tiene. Sin embargo, este concepto es independiente de cómo se representen los Objetos Administrados, así como de los tipos de permisos que puedan tener.
Analizador de Solicitudes	*Analizador de Solicitudes*	Los conceptos difieren en cuanto a los tipos de solicitudes, ya que en el modelo OSI existen siete tipos, y en el modelo de Internet existen cuatro.	*Analizador de Solicitudes*	Entidad que analiza los tipos de solicitudes que puede procesar el Agente de acuerdo a un Protocolo de Administración de Red.

Conceptos		Diferencias entre Conceptos	Concepto Genérico	Descripción del Concepto Genérico
Modelo OSI	Modelo de Internet			
Base de Información de Administración	*Base de Información de Administración*	La principal diferencia entre los conceptos es que la MIB del modelo OSI es mucho más compleja respecto a la del modelo de Internet, ya que el modelo OSI maneja bitácoras, discriminadores de envío de eventos, y el modelo de Internet no, además en el modelo OSI los Objetos Administrados se representan como tal (objetos), mientras que en el modelo de Internet los Objetos se pueden representar como atributos.	*Base de Información de Administración*	Encargada del almacenamiento y administración de los Objetos Administrados del Agente sin importar su representación y procesos intermedios que permitan generar notificaciones y reportes al Agente.
Buscador de Objetos Administrados	*Buscador de Objetos Administrados*	La diferencia entre estos conceptos radica en el tipo de estructura que es utilizada para la ubicación de Objetos Administrados, lo cual indica que utilizan diferentes algoritmos para la localización de Objetos.	*Buscador de Objetos Administrados*	Entidad que realiza búsquedas de Objetos Administrados, la cual permite buscar Objetos Administrados sin importar el tipo de organización de los mismos, ni el algoritmo utilizado para su búsqueda.
Buscador de Recursos	*Buscador de Recursos*	Dado que es un elemento de la Administración Virtual de Hardware, y ésta es independiente del modelo de información, no tienen diferencia entre ellos, a menos que se utilicen diferentes algoritmos de búsqueda.	*Buscador de Recursos*	Entidad que realiza búsquedas de Recursos, sin importar el tipo de algoritmo utilizado para la búsqueda.
Discriminador de Envío de Eventos	*Filtro*	La diferencia entre éstos conceptos es que el Discriminador de Envío de Eventos es más complejo que el Filtro, ya que éste analiza las causas por las cuales se presenta el cambio de estado mientras que el filtro solamente analiza los límites frontera del Objeto Administrado que cambió.	*Filtro de Envío de Eventos*	Entidad que permite discriminar cualquier tipo de eventos que pueden ser enviados al Administrador.

Conceptos		Diferencias entre Conceptos	Concepto Genérico	Descripción del Concepto Genérico
Modelo OSI	Modelo de Internet			
Lista de Espera de Solicitudes	*Lista de Espera de Solicitudes*	La única diferencia entre éstos conceptos son el tipo de solicitudes que se almacenan dentro de ellas.	*Lista de Espera de Solicitudes*	Entidad que almacena las solicitudes hechas por el Administrador. Esta entidad es independiente del tipo de solicitudes que puede almacenar.
Lista de Identificadores de Solicitud	*Lista de Identificadores de Solicitud*	En este caso la única diferencia que existe entre estos conceptos es la representación del identificador de solicitud que almacenan.	*Lista de Identificadores de Solicitud*	Entidad que almacena los identificadores de solicitudes enviadas por parte del Administrador. Esta entidad es independiente de la representación del identificador de solicitud.
Lista de Mapeo de Recursos	*Lista de Mapeo de Recursos*	La única diferencia que existe entre estos conceptos es la representación del identificador de Objeto Administrado utilizado para mapear recursos.	*Lista de Mapeo de Recursos*	Entidad que almacena las relaciones entre Objetos Administrados y Recursos. Esta entidad es independiente de la representación de los identificadores de Objetos Administrados utilizados en el mapeo.
Mensaje	*Mensaje*	La diferencia entre los mensajes es su representación, ya que en el modelo OSI los mensajes se representan sobre la base del protocolo CMIP y en el modelo de Internet sobre SNMP.	*Mensaje*	Información que determina las acciones que el Administrador desea llevar a en el Agente. Este concepto es independiente del tipo de acciones que se deseen realizar, así como de su representación.
Notificación	*Notificación*	La principal diferencia entre estos conceptos es su representación.	*Notificación*	Información que representa el resultado de acciones realizadas en los Objetos Administrados.
Objeto Administrado	*Objeto Administrado*	La diferencia reside en su representación, ya que un Objeto Administrado de Internet parece un atributo de un Objeto Administrado de OSI. Los Objetos Administrados del modelo OSI interpretan acciones, los del modelo de Internet no.	*Objeto Administrado*	Estructura que representa a un Recurso de un Elemento de Red, el cual es independiente de su representación.

Conceptos		Diferencias entre Conceptos	Concepto Genérico	Descripción del Concepto Genérico
Modelo OSI	Modelo de Internet			
Proceso	*Proceso*	No existe diferencia ya que se requiere para ejecutar los comandos del protocolo de Administración de Red.	*Proceso*	Mecanismo que permite ejecutar y cancelar acciones sobre Objetos Administrados. Este mecanismo puede o no ser utilizado en el dominio.
Recurso	*Recurso*	Dado que un recurso se encuentra en el ámbito de la AVH, no existe diferencia entre ellos.	*Recurso*	Entidad que forma parte de un Elemento de Red, el cual brinda parte de los servicios del Elemento de Red.
Reporte de Cambio de Estado	*Reporte de Cambio de Estado*	Esta información se encuentra en el ámbito de la AVH, por lo cual no existe una diferencia entre ellos.	*Reporte de Cambio de Estado*	Información que representa el estado de un recurso al momento en que ocurrió algún evento.
Servicio	*Servicio*	Los servicios son aquellas funcionalidades definidas en los Agentes. La diferencia entre los modelos es que los servicios soportan a los comandos de los protocolos de Administración de Red, los cuales son muy diferentes entre sí.	*Servicio*	Funcionalidad definida en el Agente. El servicio da soporte a los comandos definidos en el protocolo de Administración de Red.
Analizador de Severidad de Alarma	*"Concepto no existente"*	En el modelo OSI existe, el cual permite analizar el tipo de alarma generada, y determinar el registro de severidad de alarma correspondiente. Esta entidad no existe en el modelo de Internet.	*Servicio Analizador de Severidad de Alarma*	Servicio que permite analizar el tipo de alarma generada, y determinar el registro de severidad de alarma correspondiente. Este servicio puede ser utilizado o no.
Bitácora	*"Concepto no existente"*	En el modelo OSI existe la entidad denominada "Bitácora", sin embargo en el modelo de Internet, no existe una entidad parecida.	*Servicio de Bitácora*	Servicio que puede ser añadido en la capa MIB, el cual permite almacenar eventos que se han generado en los recursos.
Filtro de Procesamiento de Bitácora	*"Concepto no existente"*	En el modelo OSI ésta entidad es utilizada para filtrar eventos que pueden ser almacenados en Bitácora. En el modelo de Internet no existe dicha entidad, por	*Servicio de Filtro de Procesamiento de Bitácora*	Entidad que permite filtrar los eventos que pueden ser almacenados en la Bitácora. Este concepto puede ser utilizado o no en el dominio.

Conceptos		Diferencias entre Conceptos	Concepto Genérico	Descripción del Concepto Genérico
Modelo OSI	Modelo de Internet			
		lo cual este concepto no existe en este modelo.		
Solicitud	*Solicitud*	La principal diferencia entre las solicitudes que generan Administradores depende de los comandos definidos en el protocolo de Administración de Red.	*Solicitud*	Evento generado por el Administrador al querer ejecutar acciones sobre Objetos Administrados pertenecientes a un Agente.
Severidad de Alarma	*"Concepto no existente"*	En el modelo de Internet no existen registros que puedan identificar la severidad de una Alarma mientras que en el modelo OSI sí.	*Severidad de Alarma*	Información que permite determinar la severidad de una alarma. Este concepto puede ser utilizado o no en el dominio.

Tabla 4.3 Abstracción Conceptual del Dominio.

4.11.2 Asociaciones entre los Conceptos del Dominio

Identificados los conceptos del dominio, se determinan las relaciones entre éstos. [Larman, 1999] sugiere una lista de categorías para encontrar asociaciones entre los conceptos. La Tabla 4.4 muestra las asociaciones entre los conceptos del dominio.

Asociación	Concepto A	Concepto B
A genera a B	Administrador	Mensaje
	Agente	Mensaje
	Administrador de Procesos	Proceso
	Objeto Administrado	Notificación
	Recurso	Reporte de Cambio de Estado
A usa a B	Agente	Analizador de Solicitudes
	Agente	Lista de Identificadores de Solicitud
	Agente	Lista de Espera de Solicitudes
	Agente	Filtro de Envío de Eventos
	Agente	Administrador de Procesos
	Lista de Identificadores de Solicitud	Solicitud
	Servicio	Proceso
	MIB	Analizador de Permisos
	MIB	Buscador de Objetos Administrados
	MIB	Servicio de Bitácora
	MIB	Lista de Mapeo de Recursos
	Objeto Administrado	Servicio Analizador de Severidad de Alarma
	Servicio de Bitácora	Filtro de Procesamiento de Bitácora
	Administrador Virtual de Hardware	Buscador de Recursos
A se comunica con B (Bidireccional)	Administrador	Agente
	Agente	MIB
	MIB	Administrador Virtual de Hardware
A contiene a B	Lista de Espera de Solicitudes	Solicitud
	MIB	Objeto Administrado
	Servicio Analizador de Severidad de Alarma	Severidad de Alarma
	Servicio de Bitácora	Objeto Administrado
A analiza a B	Analizador de Solicitudes	Solicitud
	Filtro de Envío de Eventos	Notificación
	Filtro de Procesamiento de Bitácora	Notificación
A busca a B	Buscador de Objetos Administrados	Objeto Administrado
	Buscador de Recursos	Recurso
A monitorea a B	Administrador	Agente
	Administrador Virtual de Hardware	Recurso

Tabla 4.4 Asociaciones entre los Conceptos del Dominio.

4.11.3 Esquematización del MCEOD

Una vez que se tienen definidos tanto los conceptos del dominio como las asociaciones entre ellos, se puede generar el Modelo Conceptual Estático de Objetos de Dominio (MCEOD).

El diagrama plasma gráficamente los conceptos así como las relaciones entre éstos. Estos elementos son puestos en el diagrama de la siguiente forma:

- Para los conceptos del dominio, se utiliza un rectángulo para cada concepto, donde en su interior se encuentra el nombre del concepto.
- Las asociaciones se representan con una línea que une a los conceptos. Esta línea representa una relación unidireccional (Línea con flecha.) o bidireccional (Línea sin flecha.). En el centro de esta línea se pone una etiqueta que permite leer la asociación que tiene un concepto respecto a otro. Por último se encuentra adyacente a cada concepto un atributo de multiplicidad que sirve para definir cuantas instancias de un concepto pueden asociarse al concepto que tiene una asociación con él.

La Figura 4.9 muestra el diagrama del MCEOD.

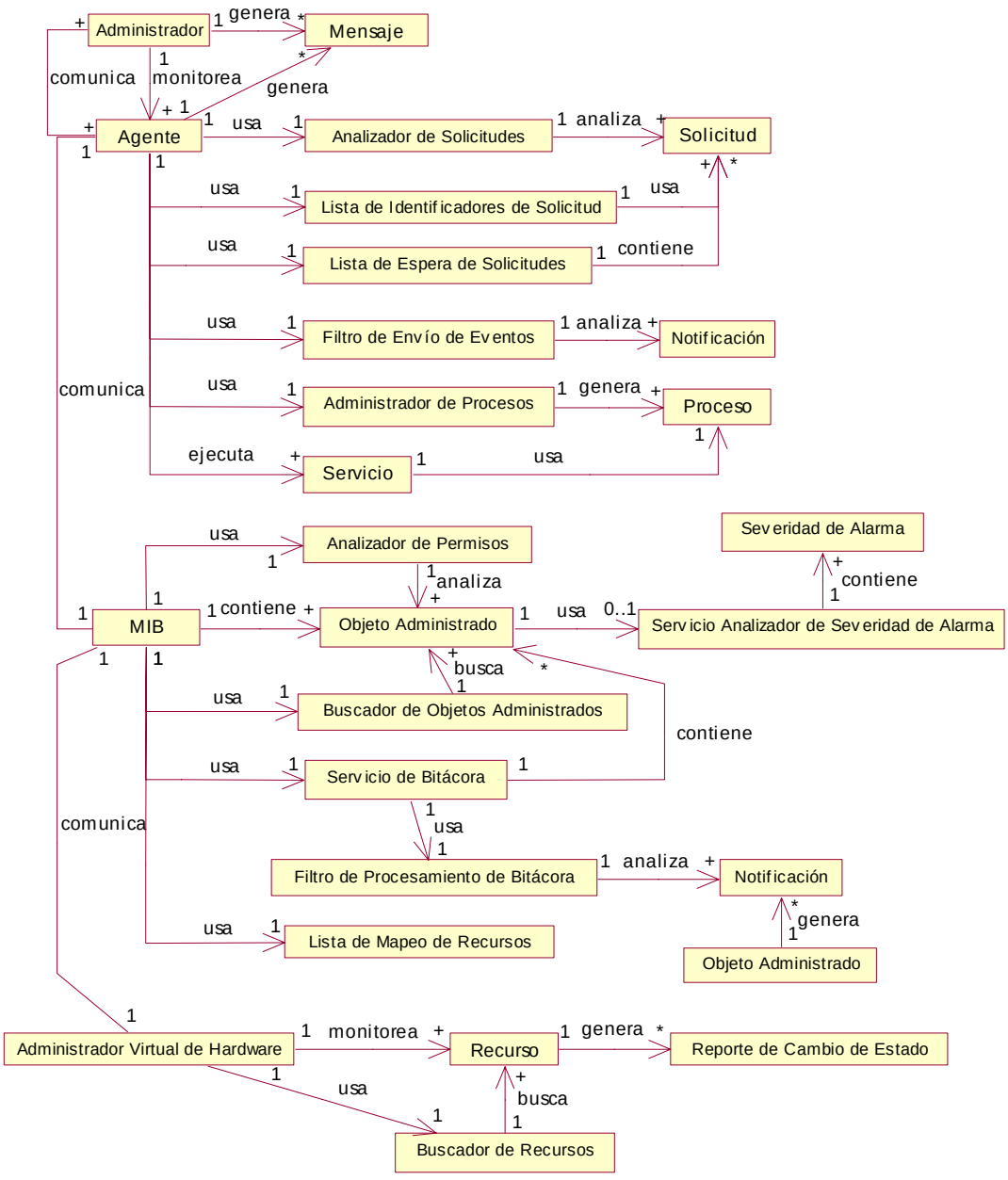


Figura 4.9 Modelo Conceptual Estático de Objetos de Dominio.

4.12 Modelo Conceptual Dinámico de Objetos de Dominio

La última etapa en el ADOO es la generación del Modelo Conceptual Dinámico de Objetos de Dominio (MCDOD), el cual permite ver el comportamiento del dominio mediante los casos de uso del dominio.

En el MCDOD intervienen los conceptos y acciones reflejados en los casos de uso del dominio (Véase el "*Anexo A*" en su sección "*A.6 Casos Abstractos de Uso del Dominio*".). Cabe destacar nuevamente, que no se trata de un modelado de software, más bien de conceptos que intervienen en el dominio (no obstante que el dominio esté ligado estrechamente con el software).

El MCDOD está compuesto de una esquematización de cada caso de uso del dominio. Al igual que en la Abstracción Vertical y Abstracción Horizontal, se utilizan diagramas de colaboración, en donde cada nodo es una instancia de un concepto.

La Figura 4.10a y Figura 4.10b muestran la esquematización del MCDOD para los casos abstractos de uso del dominio y "*Reportar Eventos generados por los Recursos*" y "*Ejecutar un Servicio en el Agente*" respectivamente.

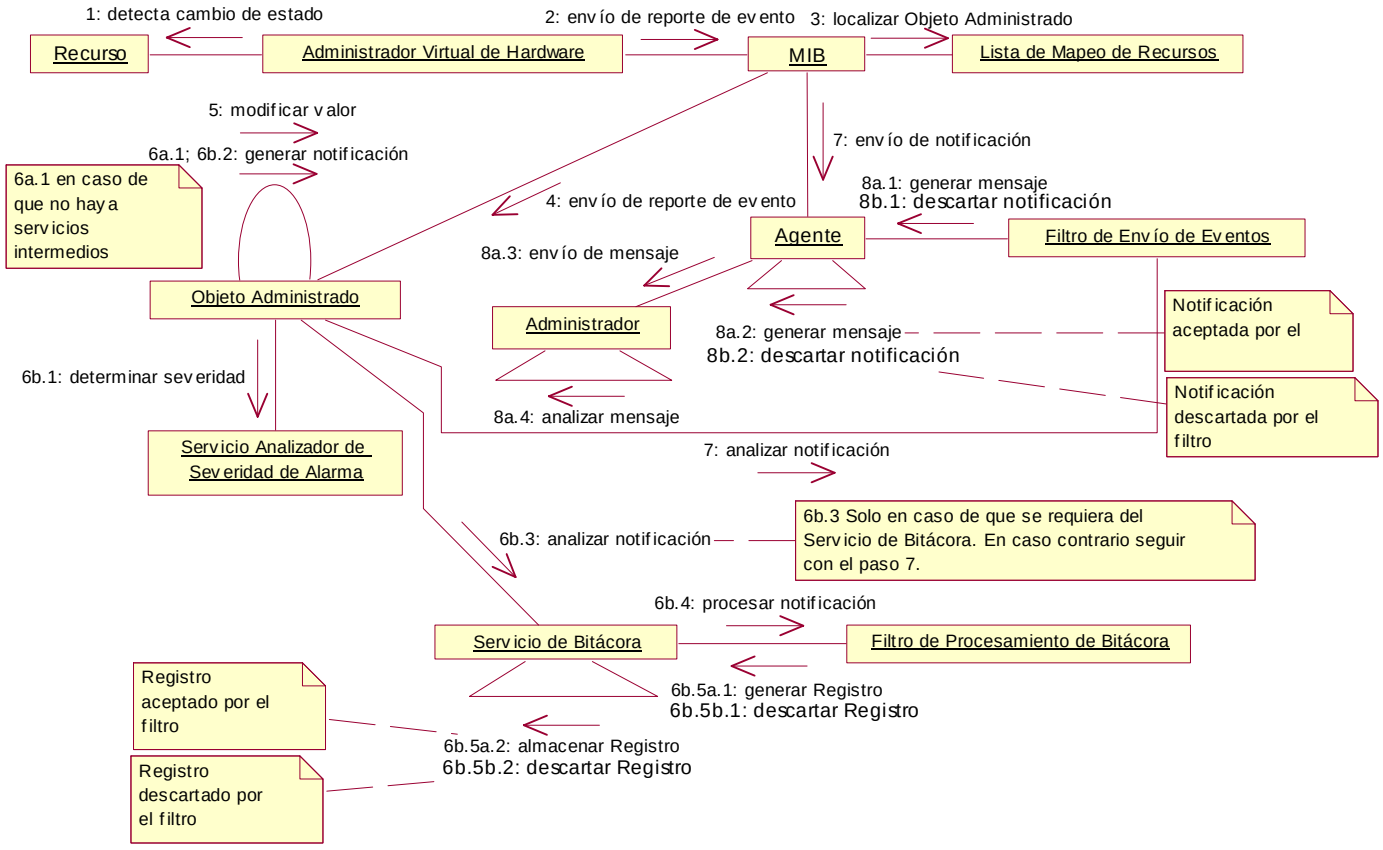


Figura 4.10a Modelo Conceptual Dinámico de Objetos de Dominio.
Caso de Uso "*Reportar eventos generados por los Recursos*".

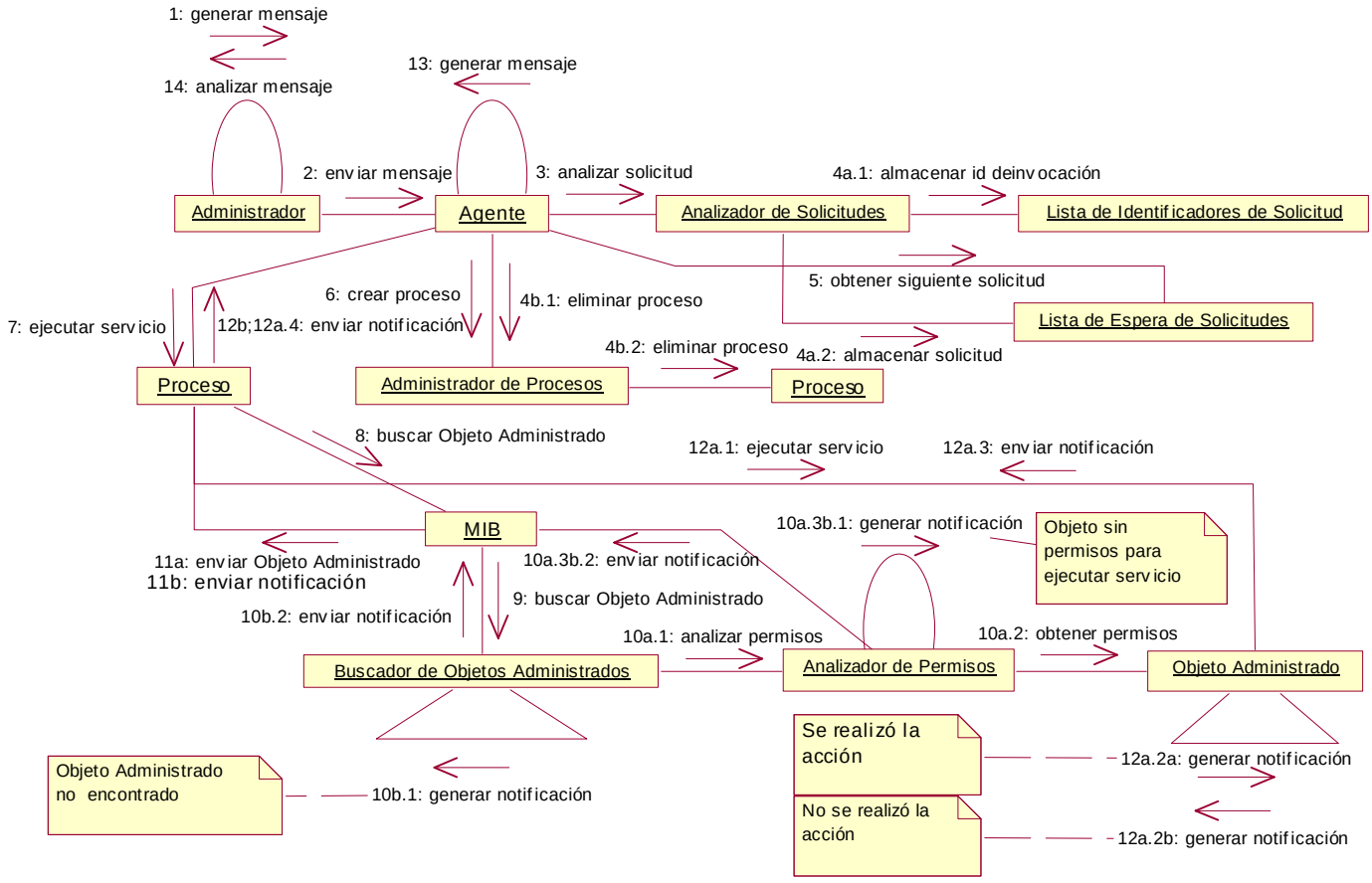


Figura 4.10b Modelo Conceptual Dinámico de Objetos de Dominio.
Caso de Uso "*Ejecutar un Servicio en el Agente*".

Ahora que se tienen determinados tanto los conceptos genéricos como las interacciones entre ellos, el dominio del framework se encuentra ya delimitado y suficientemente comprendido como para continuar con la siguiente etapa del desarrollo de un framework, el Análisis de Variabilidad en los Componentes del Dominio (Véase el "*Capítulo 5. Análisis de Variabilidad en los Componentes del Dominio*".), la cual es propuesta por la presente tesis.

4.13 Conclusiones

En el presente capítulo se tienen las siguientes conclusiones:

- El dominio del framework a desarrollar está estrechamente delimitado por la función de un Agente de Administración de Red, el cual interpreta y ejecuta servicios en representación del Administrador.

- Los casos de uso no abstractos no permiten definir los requerimientos de un framework, debido a que se aplican específicamente a un subdominio o servicio en particular. Por lo cual es necesario abstraerlos de una forma tal que permita obtener conceptos y acciones genéricas útiles para el framework.

- La Abstracción Vertical es un mecanismo que permite abstraer casos de uso similares entre sí dado un subdominio.

- La Abstracción Horizontal es un mecanismo que permite abstraer casos abstractos de uso de los subdominios, para entonces tener casos abstractos de más alto nivel, los cuales representan los requerimientos del framework.

- El Modelo Conceptual Estático de Objetos de Dominio permite observar los conceptos genéricos y las relaciones que existen entre éstos. Este modelo es la base para poder diseñar elementos de software que los puedan representar.

- El Modelo Conceptual Dinámico de Objetos de Dominio permite observar instancias de conceptos genéricos y las interacciones entre éstos. Este modelo es la base para poder determinar los servicios y funciones que cada elemento de software deberá proporcionar.

- El Análisis de Dominio Orientado a Objetos solo toma en cuenta conceptos, por lo cual es necesario mapear los conceptos y acciones en clases, objetos y métodos para poder diseñar el framework.

Capítulo

5

Análisis de Variabilidad en los Componentes del Dominio

5.1 Introducción

Esta etapa es una propuesta de la presente tesis, la cual consiste en realizar un análisis intermedio entre el Análisis de Diseño Orientado a Objetos (ADOO) y el Diseño de Dominio Orientado a Objetos, el cual sea útil en el desarrollo del framework.

La principal función de esta etapa es la de presentar, los diferentes componentes de alto nivel en el dominio lo cual permita una clasificación de los diferentes servicios que brindará el framework.

Para cada componente existen varios aspectos comunes y variables que deberán de describirse para que puedan servir de base en las decisiones que se deban de tomar en el diseño del framework.

Por último se presentan las problemáticas que se detectaron en las distintas variabilidades de cada componente, las cuales serán resueltas en el diseño de los frameworks, dependiendo de las reglas de diseño en cada uno de ellos.

Como se puede observar, esta etapa es fundamental para la detección de adaptaciones y mecanismos internos que se deberán de diseñar en el framework, lo cual permita un soporte adecuado a las diferentes aplicaciones que se puedan desarrollar.

5.2 Definición

El Análisis de Variabilidad en los Componentes del Dominio (AVCD) consiste de la definición y descripción de aquellas características tanto variables como comunes de los componentes de alto nivel del dominio, además de las posibles problemáticas que se pudieran encontrar al tratar de incorporar dichas características en el diseño. Entiéndase como componentes del dominio aquellas entidades que incorporan un conjunto de servicios relacionados que pueden involucrar la participación e interoperabilidad de muchos objetos.

Las características comunes de los componentes del dominio en la presente tesis se les denominará como Puntos Comunes en los Componentes del Dominio, así como a las características variables de los componentes del dominio se les denominará Puntos de Variabilidad en los Componentes del Dominio.

5.3 Descripción de la Fase de Análisis de Variabilidad en los Componentes del Dominio

La fase del AVCD se compone de una serie de actividades para definir los componentes del dominio así como los puntos variables y estáticos en cada uno de ellos. La Figura 5.1 nos muestra gráficamente esta fase en términos de sus actividades y sus entregables.

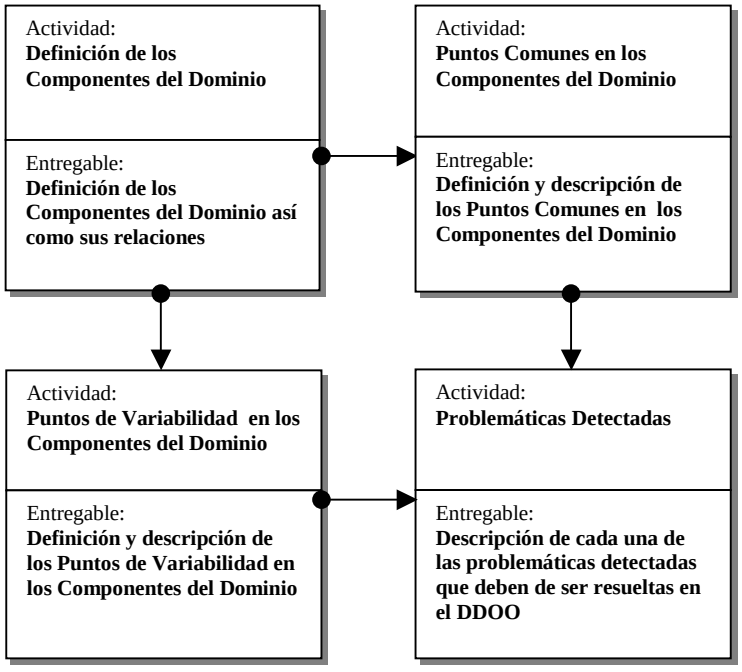


Figura 5.1 Proceso de Análisis de Variabilidad en los Componentes del Dominio.

Las actividades a desarrollar son las siguientes:

- **Definición de los Componentes del Dominio**.
  En esta actividad, se describen los componentes que pertenecen al dominio, los cuales no son mas que una entidad que representa un conjunto de servicios relacionados que dan soporte al dominio.
  Entregable
  El entregable en esta actividad es una descripción general de los componentes del dominio así como las relaciones que se tienen entre ellos.

- **Puntos Comunes en los Componentes del Dominio**.
  En esta actividad se definen aquellas características que no varían en cada uno de los componentes del dominio, independientemente del diseño o la implementación de los servicios que proveen. En este caso, se utilizan los conceptos que estén directamente relacionados o que pertenezcan a cada componente, y se define para cada uno de ellos las características comunes, es decir, aquellas que no cambian independientemente del subdominio al que se aplique el concepto.
  Entregable
  El entregable en esta actividad es la descripción de los puntos estáticos descubiertos en cada uno de los componentes del dominio.

- **Puntos de Variabilidad en los Componentes del Dominio**.
  En esta actividad se definen aquellas características que varían en cada uno de los componentes del dominio, tanto en el diseño como en la implementación de los servicios que proveen. Estos puntos variables permiten identificar la extensibilidad del framework. En este caso, se utilizan los conceptos que estén directamente relacionados o que pertenezcan a cada componente, y se define para cada uno de ellos las características que varían entre los subdominios.
  Entregable
  El entregable en esta actividad es la descripción de los puntos de variabilidad descubiertos en cada uno de los componentes del dominio.

- **Problemáticas Detectadas**.
  En esta actividad, se analizan y detectan aquellas problemáticas que las diferentes variabilidades en los componentes del dominio pudieran repercutir en el diseño del framework, las cuales deberán de ser resueltas en dicha etapa de desarrollo.
  Entregable
  El entregable en esta actividad es la serie de problemáticas que han sido detectadas basándose en las variabilidades de cada uno de los componentes del dominio.

5.4 Definición de los Componentes del Dominio

Anticipadamente, los Componentes de Alto Nivel en el Dominio fueron detectados en la etapa del ADOO en su actividad de Definición del Dominio (Véase la "*Figura 4.4. Diagrama de Demarcación del Dominio*" en el "*Capítulo 4. Análisis de Dominio Orientado a Objetos*"), donde se encuentran a tres entidades que tienen un conjunto de servicios relacionados que sirven como apoyo a otras entidades, dichos componentes son:

- **Agente**. Encargado de enviar los reportes de cambio de estado de los objetos administrados por parte del Agente a su Administrador, así como también de recibir y ejecutar solicitudes de información de parte del Administrador al Agente. Se relaciona con la MIB en el aspecto de que se ocupa de enviar notificaciones al Administrador, así como también le hace llegar las solicitudes de información que requiere el Administrador.

- **MIB**. Contiene el conjunto de objetos administrados que representan a los recursos de un elemento de red. Su relación con la Administración Virtual de Hardware es la de aceptar todos los reportes de cambio de estado en los recursos y reflejar dichos cambios en los objetos administrados. Se relaciona también con el Agente, ya que necesita de sus servicios para poder enviar las notificaciones pertinentes de cambio de estado al Administrador.

- **Administración Virtual de Hardware**. Encargada de monitorear los cambios de estado de los Recursos. Su relación con la MIB es el reporte de estos cambios de estado para que sean reflejados posteriormente en los objetos administrados, así como la aceptación de solicitudes de cambios de estado en las variables de los recursos.

Como se puede observar, cada uno de estos componentes de alto nivel, puede involucrar la participación de muchos objetos tanto para poder brindar los servicios de soporte a otros componentes, como para poder solicitar servicios.

Cada componente de alto nivel tiene sus aspectos variables así como aspectos estáticos que están íntimamente relacionados con el tipo de diseño e implementación del Agente que se desea desarrollar, dichos aspectos son analizados en las posteriores secciones en cada uno de los componentes de alto nivel en el dominio.

5.5 Puntos Comunes y Variables en el Componente "Agente"

En esta etapa, se toman como entrada los conceptos genéricos obtenidos en la etapa del ADOO que pertenecen o están directamente relacionados con la capa del Agente. La Tabla 5.1 ilustra las los puntos comunes y variables de cada concepto.

Concepto	Características Comunes	Características Variables
Administrador de Procesos	• Servicio común entre Agentes independiente del modelo de información. • Administración de Procesos. ° Creación. ° Eliminación.	• Generación de Procesos. ° Sobre demanda. ° Pool de Procesos. • Eliminación de Procesos. ° Lógica. ° Física.
Agente	• Servicio de Interpretación y Ejecución de Comandos.	• Protocolo de Administración de Red. ° CMIP. ° SNMP. • Servicios antes de interpretar o enviar un mensaje. ° Servicio de Encriptación. ° Servicio de Autenticación. ° Servicio de Compactación.
Analizador de Solicitudes	• Servicio común en los Agentes, encargado de analizar las solicitudes del Administrador.	• Diversidad de solicitudes a interpretar. ° Solicitudes CMIP. ° Solicitudes SNMP. • Formato de Solicitudes e Información de Solicitud. ° Formato CMIP. ° Formato SNMP.
Filtro de Envío de Eventos	• Filtro común en los Agentes, utilizado siempre antes de enviar una notificación de cambio de estado en un Recurso.	• Diversidad de notificaciones. ° Notificaciones CMIP. ° Notificaciones SNMP.
Lista de Espera de Solicitudes	• Contenedor común en los Agentes, para evitar la pérdida de solicitudes.	• Obtención del mensaje siguiente que debe interpretar el Agente ° Por prioridad del Mensaje. ° Secuencial.
Lista de Identificadores de Solicitud	• Contenedor común en los Agentes, para almacenar identificadores de solicitud.	• Formato de Identificadores de Solicitud. ° Formato CMIP. ° Formato SNMP. • Discriminantes utilizados para permitir o negar el envío de reportes de cambio de estado

Concepto	Características Comunes	Características Variables
Mensaje	• Información intercambiada entre Administradores y Agentes.	• Diversidad de Mensajes. ° Mensajes CMIP. ° Mensajes SNMP. • Formato de Mensaje. ° Formato CMIP. ° Formato SNMP.
Notificación	• Información utilizada entre elementos de la MIB y el Agente para determinar el estado final de un servicio.	• Diversidad de notificaciones. ° Notificación CMIP. ° Notificación SNMP.
Proceso	• Utilizado para la ejecución de un servicio.	• Ninguna.
Servicio	• Elemento que permite la ejecución de una solicitud en particular.	• Diversidad de servicios. ° Servicio CMIP. ° Servicio SNMP. • Modo. ° Confirmado. ° No Confirmado. • Tipo. ° Atómico. ° Mejor esfuerzo.
Solicitud	• Elemento que contiene información sobre el servicio que se debe ejecutar en el Agente.	• Diversidad de solicitudes. ° Solicitud CMIP. ° Solicitud SNMP.

Tabla 5.1 Características Comunes y Variables en los Conceptos de la Capa Agente.

Todas estas características se obtuvieron principalmente de las fuentes de conocimiento, y casos de uso.

5.6 Puntos Comunes y Variables en el Componente "MIB"

En la presente etapa, se toman como entrada los conceptos genéricos obtenidos en la etapa del ADOO que pertenecen o están directamente relacionados con la capa MIB. La Tabla 5.2 ilustra las los puntos comunes y variables de cada concepto.

Concepto	Características Comunes	Características Variables
Analizador de Permisos	• Servicio común en MIBs, ya que verifica los permisos de acción sobre un Objeto Administrado.	• Tipos de Permisos a analizar. ° Lectura. ° Escritura.
Buscador de Objetos Administrados	• Servicio común en MIBs, que permite buscar Objetos Administrados.	• Identificadores de Búsqueda. ° Identificadores CMIP. ° Identificadores SNMP. • Información de Búsqueda.
Filtro de Procesamiento de Bitácora	• Ninguna.	• Presencia. ° Presente en Agentes CMIP. ° No presente en Agentes SNMP. • Diversidad de notificaciones a analizar.
Objeto Administrado	• Elemento común en MIBs, el cual contiene información relevante sobre un Recurso. • Identificador único.	• Diversidad de Clases de Objetos Administrados. • Representación ° Representación CMIP. ° Representación SNMP.
Notificación	• Información utilizada entre elementos de la MIB y el Agente para determinar el estado final de un servicio.	• Diversidad de notificaciones. ° Notificación CMIP. ° Notificación SNMP.
MIB	• Contenedor de Objetos Administrados común en los Agentes. • Modelo Jerárquico de Contención.	• Diversidad de Objetos Administrados a contener. • Contenedor. ° Base de Datos. ° Archivo. ° Memoria.
Servicio Analizador de Severidad de Alarma	• Ninguna.	• Presencia. ° Presente en Agentes CMIP. ° No existe en Agentes SNMP. • Diversidad de notificaciones a analizar.
Servicio de Bitácora	• Ninguna.	• Presencia. ° Presente en Agentes CMIP. ° No presente en Agentes SNMP.
Severidad de Alarma	• Ninguna.	• Presencia. ° Presente en Agentes CMIP. ° No presente en Agentes SNMP.

Tabla 5.2 Características Comunes y Variables en los Conceptos de la Capa MIB.

5.7 Puntos Comunes y Variables en el Componente "AVH"

En esta etapa, se toman como entrada los conceptos genéricos obtenidos en la etapa del ADOO que pertenecen o están directamente relacionados con la capa de la Administración Virtual de Hardware. La Tabla 5.3 ilustra las los puntos comunes y variables de cada concepto.

Concepto	Características Comunes	Características Variables
Administrador Virtual de Hardware	• Servicio común a cualquier Agente, el cual permite una interfaz entre acciones en Objetos Administrados y Recursos físicos.	• Monitoreo. ° Round Robin. ° Por prioridad. ° Secuencial. • Diversidad de Recursos a monitorear.
Buscador de Recursos	• Servicio común en la AVH, que permite buscar Recursos.	• Diversidad de Identificadores de Búsqueda. • Información de Búsqueda.
Lista de Mapeo de Recursos	• Contenedor utilizado para mapear un Recurso a un Objeto Administrado y viceversa.	• Ninguno
Recurso	• Elemento presente en cualquier AVH, el cual tiene la interfaz necesaria para ejecutar una acción en un Recurso físico.	• Diversidad de Recursos.
Reporte de Cambio de Estado	• Información común en la AVH, la cual contiene un reporte de cambio de estado en un Recurso.	• Formato de Reporte.

Tabla 5.3 Características Comunes y Variables en los Conceptos de la Capa AVH.

5.8 Problemáticas Detectadas

En esta última actividad del AVCD, se detectan las diferentes problemáticas que pueden repercutir en el diseño del framework, dependiendo de los Puntos de Variabilidad en los Componentes del Dominio.

Las problemáticas que se deben de resolver en el diseño del framework en cada uno de los componentes de alto nivel en el dominio son las siguientes:

5.8.1 Agente

- Se deben de abstraer las diferentes similitudes entre los dos protocolos de administración de red, lo cual permita de manera simple ajustar ya sea uno u otro protocolo al framework.

- Debe existir un mecanismo que permita configurar los servicios que se requieran utilizar antes de enviar un mensaje o recibirlo, como lo son servicios de autenticación, encriptación, etc.

5.8.2 MIB

- Se deben encontrar diferentes mecanismos para poder ajustar el framework dependiendo de los diferentes Objetos Administrados que se requieran mantener en la MIB para efecto de administración en el Elemento de Red.
- Se debe de ajustar la representación de los Objetos Administrados dependiendo del modelo de administración que se requiera (ya sea Internet u OSI) o encontrar un mecanismo que pueda soportar a ambos modelos sin necesidad de muchas adaptaciones.
- Dependiendo del modelo de administración, debe existir una manera de eliminar o incluir mecanismos como los Filtros, la Bitácora, y el Discriminador de Envío de Eventos entre otros servicios.
- Deben diseñarse mecanismos que permitan que los dos modelos de administración puedan satisfactoriamente generar notificaciones genéricas, sin importar el modelo utilizado.

5.8.3 Administración Virtual de Hardware

- Se deben encontrar diferentes mecanismos para poder ajustar el framework dependiendo de los diferentes Recursos que se tengan en el Elemento de Red que utilice al framework, además de las diferentes clases de monitoreo.

5.8 Conclusiones

En el presente capítulo se tienen las siguientes conclusiones:

- Esta fase del AVCD permite tener un panorama más claro de las diferentes problemáticas que se deberán de resolver en el diseño del framework.

- La principal función de esta fase es la de describir tanto los componentes de alto nivel, lo que permite una mejor separación de los diferentes servicios que proveerá el framework, así como las variabilidades de cada componente.

- Mediante el AVCD, el diseño del framework sólo se abocará a resolver las problemáticas detectadas en esta etapa, lo cual permite una aceptable separación de actividades puramente de diseño (las cuales se tratan en la siguiente fase) con actividades de análisis.

Capítulo

6

Diseño de Dominio Orientado a Objetos

6.1 Introducción

En el presente capítulo, se diseña un framework orientado a objetos, el cual sirve de base para el desarrollo de Agentes de Administración de Red. El diseño del framework es el resultado de aplicar la fase de Diseño de Dominio Orientado a Objetos (DDOO), la cual es una etapa de la Ingeniería de Dominio Orientado a Objetos (IDOO).

En el DDOO recibe como entrada los puntos comunes y los puntos de variabilidad pertenecientes a cada uno de los conceptos de dominio descritos en el Análisis de Variabilidad en los Componentes del Dominio (AVCD). Los puntos comunes y de variabilidad de cada concepto son resueltos utilizando principalmente patrones de diseño definidos en [Gamma, 1995].

Los conceptos de dominio son mapeados en clases y sus puntos comunes y de variabilidad son reflejados en sus métodos. Una vez realizado lo anterior, se diseña el mecanismo de inicialización del framework, el cual permite configurarlo y extenderlo para generar la aplicación deseada.

Las clases obtenidas en el punto anterior, se plasman en un esquema integral, el cual se denomina Modelo Estático de Objetos de Dominio (MEOD), lo cual permite ver a detalle las relaciones estáticas entre éstas.

Por último se diseña el Modelo Dinámico de Objetos de Dominio (MDOD), en donde se modela la interacción esperada entre instancias de clases definidas en el MEOD.

6.2 Definición

El DDOO es la fase de la IDOO, la cual toma como entrada los conceptos y los casos de uso de dominio obtenidos en la etapa del ADOO, y los puntos estáticos y de variabilidad de cada concepto determinados en el AVCD.

Los conceptos de dominio son mapeados en clases, y tanto los puntos comunes como los puntos de variabilidad de los conceptos son resueltos utilizando "*patrones de diseño*" [Gamma, 1995].

El DDOO determina los estándares de diseño, los cuales sirvan de guía en la definición de clases y la formación de jerarquías de herencia y composición. Además define el MEOD el cual integra las soluciones[10] a los puntos comunes y de variabilidad de cada uno de los conceptos de dominio. Por último define el MDOD donde se plasman las interacciones entre instancias de clase determinadas en el MEOD, lo que determina el comportamiento esperado en el framework y en las aplicaciones que lo extienden.

6.3 Descripción de la Fase de Diseño de Dominio Orientado a Objetos

La fase del DDOO comprende de un conjunto de actividades, las cuales están orientadas a generar el diseño del framework. Todas éstas actividades son ilustradas en la Figura 6.1.

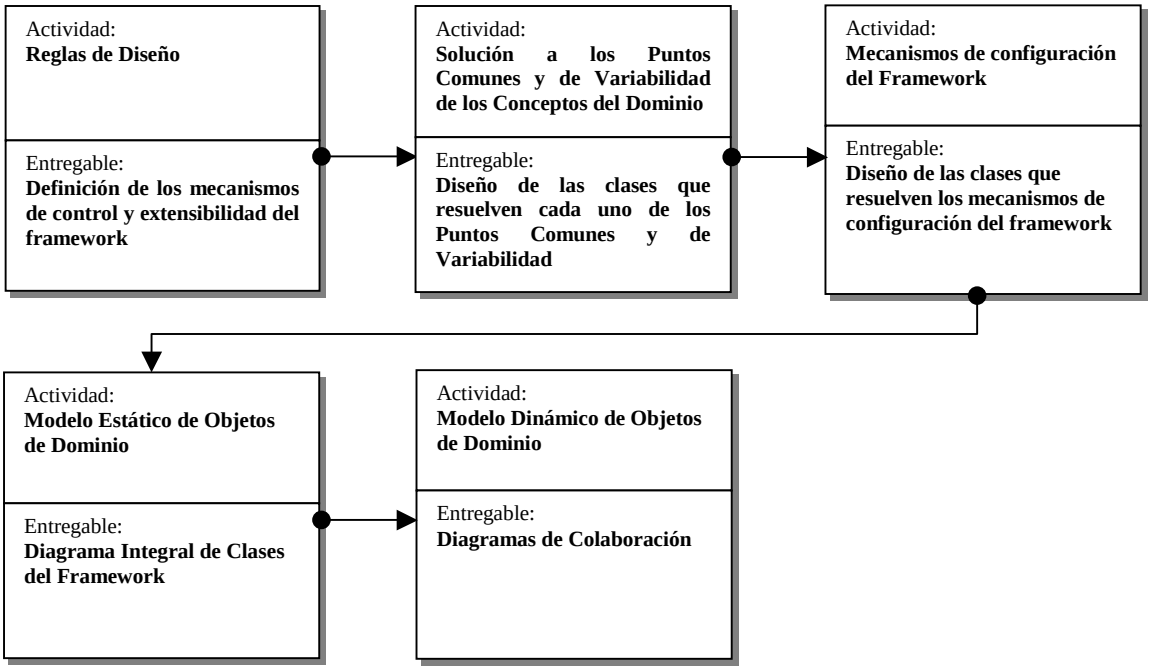


Figura 6.1 Proceso de Diseño de Dominio Orientado a Objetos.

[10] Las soluciones a los puntos comunes y de variabilidad son reflejadas en clases.

Las actividades a desarrollar son las siguientes:

- **Reglas de Diseño**
  En esta actividad, se describen principalmente aquellos mecanismos de orientación a objetos que se utilizarán para diseñar, controlar y extender el framework.
  Entregable
  El entregable en esta actividad es una descripción de los mecanismos de control y extensibilidad del framework.

- **Solución a los Puntos Comunes y de Variabilidad en los Conceptos del Dominio**
  En esta actividad, los puntos comunes y de variabilidad en los conceptos del dominio obtenidos en la fase del AVCD, son resueltos o diseñados utilizando patrones de diseño.
  Entregable
  El entregable es el diseño de clases las cuales mapean los conceptos de dominio y sus puntos estáticos y de variabilidad, describiendo además el porqué se utilizó el patrón de diseño que los resuelve.

- **Mecanismos de Configuración del Framework**
  Esta actividad define los mecanismos de configuración que tendrá el framework para el desarrollo de aplicaciones basadas en el mismo.
  Entregable
  El entregable de esta actividad es el diseño de las clases y mecanismos de configuración necesarios para el desarrollo de aplicaciones.

- **Modelo Estático de Objetos de Dominio**.
  El MEOD muestra gráficamente de manera conjunta tanto las clases diseñadas en cada una de las actividades anteriores, como las relaciones entre éstas. Este modelo es la base para la extensibilidad de cada una de las aplicaciones.
  Entregable.
  El entregable de esta actividad es la esquematización del modelo estático de objetos de dominio.

- **Modelo Dinámico de Objetos de Dominio**.
  El MDOD muestra eventos, colaboraciones, y relaciones de tiempo entre cada una de las instancias de clases del modelo estático de objetos de dominio sobre el curso de los casos de uso del dominio. Este modelo demuestra si todas las responsabilidades han sido bien asignadas y distribuidas entre las clases.
  Entregable.
  El entregable de esta actividad es la esquematización del MDOD.

6.4 Reglas Generales de Diseño

Las reglas generales de diseño del framework determinan principalmente la notación y semántica del diseño de las clases. Además determinan la forma en que se documentará el diseño del framework, así como la forma en que éste podrá extenderse para desarrollar las aplicaciones. Las reglas de diseño son:

1. Se utilizará el lenguaje UML para el diseño de las clases.
2. El nombrado de clases estará basado en el idioma inglés.
3. Las clases y métodos abstractos reflejados en los diagramas de clase, deberán aparecer en letras itálicas, mientras que los métodos concretos deben aparecer en letras regulares.
4. El diseño de las clases tendrá una relación directa respecto de los conceptos de dominio, por lo cual al diseñar clases del framework éstas deberán estar agrupadas por concepto.
5. Además de agrupar las soluciones por concepto, éstas deberán a su vez estar agrupadas por componente de dominio, los cuales están definidos en el AVCD.
6. Cada solución de diseño propuesta a los conceptos del dominio debe ser documentada de forma que cubra los siguientes puntos:
   - *Responsabilidad*. Descripción de la responsabilidad con la que cuentan el o los conceptos.
   - *Puntos Comunes*. Descripción de los puntos comunes del o los conceptos.
   - *Puntos de Variabilidad*. Descripción de los puntos de variabilidad del o los conceptos.
   - *Solución*. Descripción de la solución de diseño aplicada a resolver los puntos estáticos y de variabilidad de los conceptos.
   - *Modelo Estático*. Diagrama de clases que corresponde a la solución de diseño, en donde se mencionan los patrones de diseño aplicados al modelo, además de una breve explicación del porqué se decidió utilizarlos.
   - *Descripción de Clases*. Descripción de cada uno de los elementos que componen a las clases presentadas en el modelo estático.
   - *Lógica de Control del Framework*. Describe los métodos que contienen la lógica de control definida en el framework, lo cual permite la aplicación del principio "*Holliwood*". Debido a esta razón esta clase de métodos no deben ser extendidos.
   - *Extensibilidad*. Guía para el desarrollador de aplicaciones en donde se explica la forma en que se deben extender clases presentadas en el modelo estático, lo cual le permita definir de manera correcta la funcionalidad esperada por el framework.
   - *Modelo Dinámico*. Diagrama de secuencia que ilustra el paso de mensajes entre las clases presentadas en el modelo estático, así como la descripción de las acciones realizadas en el mismo.
7. En caso de que varios conceptos estén muy relacionados, es decir que sea más claro documentar su diseño de solución de manera conjunta, éste se realizará de esta forma.
8. El diseño del framework deberá estar orientado principalmente a ser lo suficientemente extensible para permitir el definir nuevos servicios, los cuales tengan una correspondencia directa con los comandos del protocolo de administración de red de las aplicaciones.

9. La configuración del framework estará basada en un archivo XML, el cual determinará las clases concretas que forman parte de la aplicación deseada, por lo cual la configuración estará orientada al enlace dinámico de clases.
10. Los tipos de datos como cadenas, enteros, etc. y clases auxiliares[11], los cuales sean utilizados en el diseño de las clases, corresponderán a los definidos en el lenguaje Java.

En las siguientes secciones se presenta el diseño del framework agrupado por componente de dominio.

[11] Un ejemplo de clase auxiliar del lenguaje Java puede ser la clase Thread, ya que brinda una interfaz para definir hilos de control, sin necesidad de que el framework diseñe dicho mecanismo.

6.5 Diseño del Componente "Agente"

Antes de determinar el diseño de los conceptos relacionados con el componente "*Agente*", es necesario definir los objetivos principales con los que contará el framework para éste componente. Cabe destacar que éstos objetivos deben estar basados en el AVCD que se efectuó sobre los conceptos que se asocian con el componente.

Los objetivos de diseño que el framework debe asegurar en el componente *Agente* son los siguientes:

1. **Ejecución de Comandos**. Debe existir el soporte necesario para que el Agente interprete y ejecute comandos provenientes del Administrador, los cuales estén determinados por un protocolo de Administración de Red.
2. **Extensión de Comandos**. Los comandos pueden ser extendidos para su interpretación, es decir, si en un futuro se determina que en alguna versión adicional del protocolo de administración de red en uso existen nuevos comandos, el framework debe ser lo suficientemente flexible para soportar este cambio.
3. **Servicios Adicionales de Interpretación de Comandos**. El Agente debe ajustarse a los servicios necesarios en la interpretación y ejecución de comandos, como lo puede ser encriptación, compactación, filtrado, o cualquier otro servicio involucrado con el análisis del mensaje.
4. **Ejecución concurrente de varias solicitudes**. El Agente debe tener el soporte de manejar concurrentemente múltiples solicitudes de ejecución de comandos y envío de reportes.

Una vez determinados los objetivos específicos que el framework deberá cumplir con el diseño del componente "*Agente*", se presenta a continuación para cada concepto asociado con el componente, su responsabilidad, los puntos comunes y de variabilidad determinados en el AVCD y la solución a éstos. Cabe destacar que los conceptos presentados en esta sección se encuentran ordenados de acuerdo al orden en cómo éstos participan en el MCDOD dado el caso de uso del dominio "*Ejecución de un Servicio en el Agente*" (Véase el "*Capítulo 4. Análisis de Dominio Orientado a Objetos*" en su sección "*4.12. Modelo Conceptual Dinámico de Objetos de Dominio*").

6.5.1 Analizador de Solicitudes, Solicitud, Mensaje, Notificación

Se decidió presentar el diseño de la solución de los conceptos *Analizador de Solicitudes, Solicitud, Mensaje* y *Notificación* de manera conjunta debido a que auxilian al Agente a obtener los mensajes enviados por el Administrador o por reportes de cambio de estado de la MIB, lo cual indica que están muy relacionados entre sí y en mejor presentar su diseño de forma agrupada a individual.

6.5.1.1 Responsabilidad

- El *Analizador de Solicitudes* es el responsable de analizar los tipos de solicitud que puede procesar el Agente de acuerdo a un Protocolo de Administración de Red, además de permitir obtener los mensajes enviados por parte del Administrador.
- La *Solicitud* tiene como responsabilidad el identificar el servicio que debe ejecutarse en el Agente.
- El concepto *Mensaje* tiene la responsabilidad de encapsular la información enviada por el Administrador para la ejecución de un servicio. Cabe destacar que una *Notificación* es una especialización de un *Mensaje*, por lo que no es necesario documentarla mas que en el diseño de las clases.

6.5.1.2 Puntos Comunes

Los puntos comunes del concepto *Analizador de Solicitudes* son:

- Servicio común en Agentes de Administración de Red.
- Analiza solicitudes convirtiendo un flujo de información en un mensaje, el cual pueda ser interpretado por el Agente.

Los puntos comunes del concepto *Solicitud* son:

- Identifica el servicio que debe ejecutarse en el Agente.

Los puntos comunes del concepto *Mensaje* son:

- Encapsula la información que es intercambiada entre Agentes y Administradores.

6.5.1.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *Analizador de Solicitudes* son:

- Protocolo orientado a conexión o no conexión para la recepción de mensajes.
- Diversidad de solicitudes a interpretar, dependiendo del protocolo de administración de red utilizado por el Agente.
- Distintos formatos de mensaje que debe producir dado el flujo de información enviado por el Administrador.

Los puntos de variabilidad del concepto *Solicitud* son:

- Diversidad de tipos de solicitudes que el Administrador puede enviar, dependiendo del Protocolo de Administración de Red utilizado.

Los puntos de variabilidad del concepto *Mensaje* son:

- Diversidad de información que es intercambiada entre el Agente y el Administrador.
- Formatos de mensajes, los cuales dependen del Protocolo de Administración de Red.

6.5.1.4 Solución

El analizador de solicitudes debe ser una entidad de software que siempre esté a la escucha de flujo de información enviada de parte del Administrador y la convierta en mensajes que el Agente pueda interpretar, por lo cual se propone que el analizador sea un hilo de control (Thread) el cual lleve a cabo ésta función. En el método principal del Thread se encuentra la lógica genérica de obtención del flujo de información y su conversión a un formato de mensaje, lo cual corresponde a los "*Puntos Comunes*" para ésta entidad. La solicitud debe estar encapsulada dentro del identificador del servicio, el cual pertenece al mensaje enviado por el Administrador. No obstante, la lógica específica de transformación de un flujo de información en un mensaje, los tipos de solicitudes configuradas en el Agente y los formatos de mensaje que son entendidos por el Agente, se delegan a entidades de software abstractas, las cuales se definen por el desarrollador del Agente, ya que en éstas recaen los "*Puntos de Variabilidad*".

6.5.1.5 Modelo Estático

El modelo estático consiste en la aplicación de los patrones de diseño "*Template Method*" y "*Strategy*" [Gamma, 1995], reflejados en el diagrama de clases de la Figura 6.2.

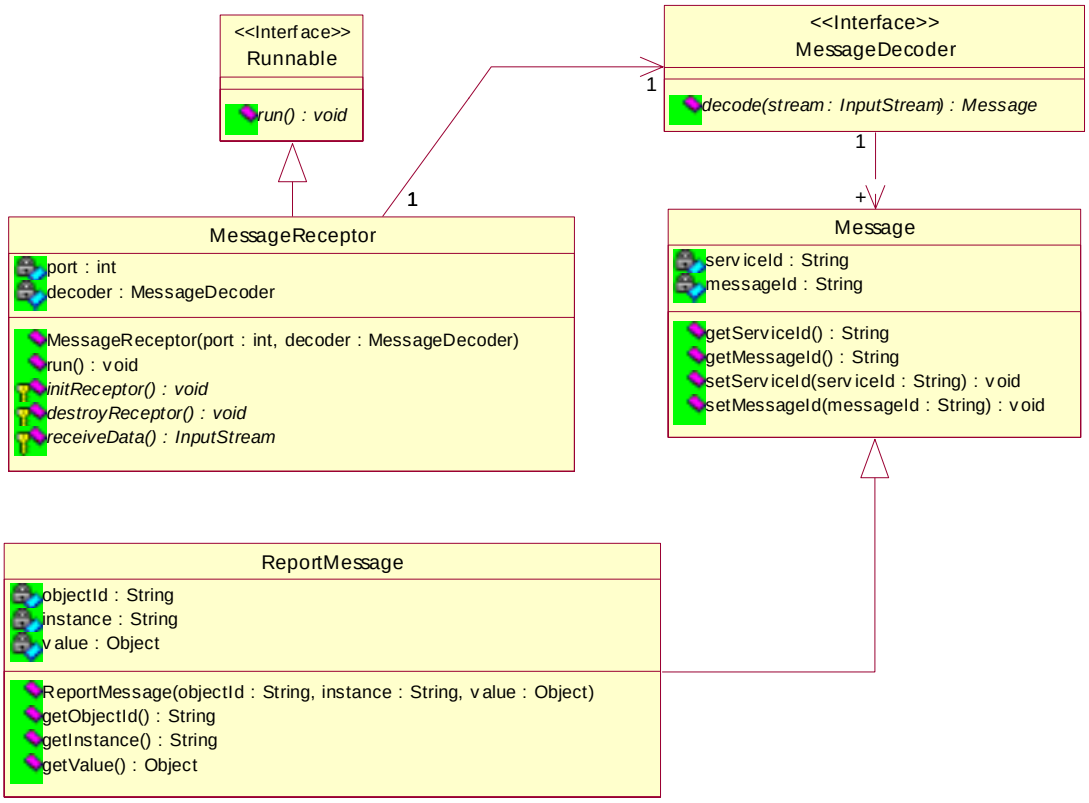


Figura 6.2 Modelo Estático de la Solución de Diseño a los Conceptos "*Analizador de Solicitudes*", "*Solicitud*", "*Mensaje*", "*Notificación*".

La aplicación de ambos patrones de diseño se describe a continuación:

- ***Template Method***. El método plantilla se encuentra en la clase "*MessageReceptor*" en su método *run()*, el cual contiene la lógica suficiente para iniciar y ejecutar el mecanismo de recepción de datos. Estos mecanismos deben ser implementados por el desarrollador del Agente a través de los métodos *initReceptor(…), receiveData(…),* y *destroyReceptor(…).*

- ***Strategy***. El método plantilla delega a un objeto de la interfaz *MessageDecoder* la responsabilidad de convertir el flujo de información enviada por el Administrador a un Mensaje, el cual pueda ser interpretado por el Agente. En este caso el desarrollador del Agente debe implementar el método *decode(…)* en una clase concreta que implemente a ésta interfaz. Cabe destacar que instancias de la interfaz *MessageDecoder* deben tener la lógica suficiente de producir instancias de subclases de *Message* lo cual permitirá que existan varios formatos de mensajes.

6.5.1.6 Descripción de Clases

La descripción de las clases *MessageReceptor, MessageDecoder, Message* se encuentra en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.2 Descripción de Clases del Componente Agente*", y la descripción de la clase *ReportMessage* se encuentra en la "*Sección C.4 Descripción de Clases del Componente AVH*".

6.5.1.7 Lógica de Control del Framework

Parte de la lógica de control del framework se encuentra reflejada en el método *run()* de la clase *MessageReceptor* ya que en él se refleja la responsabilidad de obtener los flujos de datos enviados por el Administrador, convertirlos en mensajes y almacenarlos en la cola de mensajes del Agente, realizando todo esto de manera concurrente para permitir que otros flujos de datos puedan ser atendidos. Este método por lo tanto no debe ser redefinido a menos que se requiera modificar la lógica de recepción de mensajes.

6.5.1.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *MessageReceptor*, *MessageDecoder*, y *Message*.

MessageReceptor. Clase encargada de recibir los mensajes enviados por el administrador, no obstante es abstracta debido a que no se sabe con exactitud que tipo de protocolo de administración de red es utilizado y si éste es orientado o no a conexión. Los métodos que deben ser concretizados son los siguientes:

- ***initReceptor(…)***. Inicia el mecanismo de obtención de mensajes, el cual puede ser un Socket servidor orientado o no a conexión, el cual permita recibir un flujo de datos, para después convertirlo a un Mensaje. El mecanismo debe inicializar la escucha de flujos de datos en el puerto determinado en el constructor de la clase.
- ***destroyReceptor(…)***. Finaliza el mecanismo de obtención de mensajes, en este caso el comportamiento esperado es que libere los recursos utilizados como mecanismo de obtención de flujos de datos.

- ***receiveData(…)***. Permite la recepción de un flujo de datos. El comportamiento esperado es que este método utilice el mecanismo de recepción de datos definido en *initReceptor(…)* y genere un *InputStream*, el cual pueda ser utilizado por *MessageDecoder* para generar el mensaje correpondiente.

MessageDecoder. Clase encargada de decodificar los mensajes enviados por el administrador. Esta clase es abstracta debido a que no se sabe con exactitud cuales sean los mensajes que se puedan recibir, aparte de los formatos de los mismos que puedan ser aceptados por el Agente. El método que debe ser concretizado es el siguiente:

- ***decode(…)***. Decodifica un flujo de entrada de datos. El comportamiento esperado es que este método lea el identificador de mensaje que se encuentra en el encabezado del paquete y elija la subclase de *Message* que encapsule la información enviada por el Administrador.

Message. Clase que encapsula la información relevante de la información enviada por el Administrador para la ejecución de un comando. No obstante que esta clase es concreta, de ella se deben derivar los formatos de mensajes que sean utilizados en la aplicación del Agente, ya que los miembros de clase *serviceId* y *messageId* son importantes debido a que se utilizan para identificar el comando a ejecutar cuando el Agente analiza el mensaje.

6.5.1.9 Modelo Dinámico

La Figura 6.3 presenta el modelo dinámico entre instancias *MessageReceptor, MessageDecoder* y *Message*.

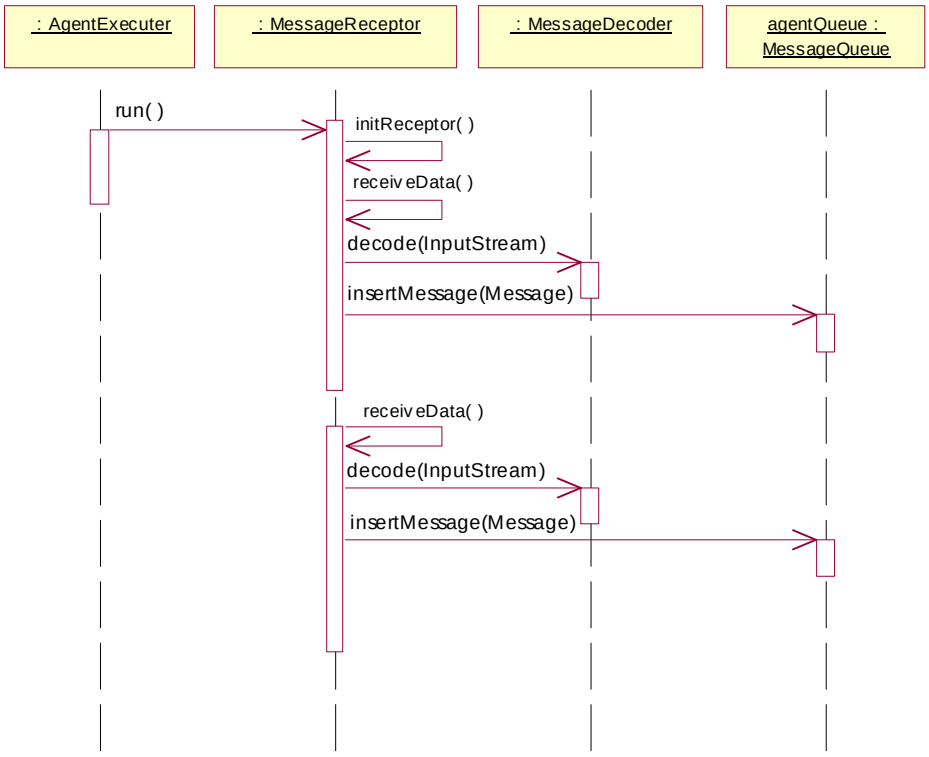


Figura 6.3 Modelo Dinámico de la Solución de Diseño a los Conceptos "*Analizador de Solicitudes*", "*Solicitud*", "*Mensaje*", "*Notificación*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. Se inicia el mecanismo de recepción de mensajes al realizar la llamada del método *initReceptor(…)* de la clases *MessageReceptor*.
2. En caso de que el Administrador envíe un flujo de datos, este es recibido a través del método *receiveData(…)* y es convertido en un flujo de entrada *InputStream*.
3. Se delega la transformación del flujo de información a *MessageDecoder* a través de su método *decode(…)*.
4. Por último se inserta el objeto *Message* en la cola de mensajes del Agente.

6.5.2 Lista de Espera de Solicitudes, Filtro de Envío de Eventos

El diseño de la solución de los conceptos *Lista de Espera de Solicitudes* y *Filtro de Envío de Eventos* se presenta de manera conjunta debido a que permiten que el Agente pueda contener temporalmente los mensajes del Administrador para la posterior ejecución de los comandos, filtrando aquellos mensajes que no deban ser ejecutados por el Agente, por lo cual es mas claro observar el diseño de ambos conceptos a observarlo de manera individual.

6.5.2.1 Responsabilidad

- La *Lista de Espera de Solicitudes* es responsable de almacenar los mensajes decodificados que fueron enviados por el Administrador, se encarga de aplicar también filtros que permiten o impiden el almacenamiento de dichos mensajes, lo cual libera al Agente de ésta responsabilidad.
- El *Filtro de Envío de Eventos* es el responsable de discriminar los reportes de cambios de estado en Objetos Administrados, de tal forma que identifique aquellos reportes que tienen que ser enviados al Administrador y aquellos que tienen que ser desechados.

6.5.2.2 Puntos Comunes

Los puntos comunes del concepto *Lista de Espera de Solicitudes* son:

- Contenedor de mensajes, lo que permite al Agente ejecutar los servicios solicitados de manera concurrente.

Los puntos comunes del concepto *Filtro de Envío de Eventos* son:

- Concepto presente en los Agentes, el cual permite discriminar reportes de cambio de estado en Objetos Administrados, y aunque varía entre cada modelo de información, su responsabilidad es la misma.

6.5.2.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *Lista de Espera de Solicitudes* son:

- Obtención del siguiente mensaje que debe analizar el Agente, ya que pueden existir mecanismo que permitan obtener mensajes ya sea por prioridad, primero en llegar primero en salir (FIFO), etc.
- No obstante que en el concepto "*Agente*" se presenta la variabilidad que consta de servicios pre-interpretación de mensajes, se decidió que ésta variabilidad sea delegada a la lista de espera de solicitudes, ya que ésta debe aplicarse antes de obtener el siguiente mensaje a analizar y esto ocurre al tratar de insertar el mensaje en la lista de espera

Los puntos de variabilidad del concepto *Filtro de Envío de Eventos* son:

- Diversidad de formatos de reportes de cambio de estado.
- Discriminantes utilizados para permitir o negar el envío de reportes de cambio de estado.

6.5.2.4 Solución

La lista de espera de solicitudes debe tener la lógica suficiente para insertar mensajes y aplicarles filtros, los cuales discriminen los mensajes y permitan o impidan su almacenamiento, uno de ellos puede ser el *Filtro de Envío de Eventos*, lo cual corresponde al "*Punto Común*" de este contenedor de mensajes.

No obstante la forma en que el Agente obtiene el siguiente mensaje en la lista de espera y los filtros que deben aplicarse al momento de intentar insertar un mensaje debe ser implementada por el desarrollador del Agente, es decir, el debe ser el encargado de concretizar los "*Puntos de Variabilidad*".

6.5.2.5 Modelo Estático

El modelo estático los conceptos *Lista de Espera de Solicitudes* y *Filtro de Envío de Eventos* consiste en la aplicación de los patrones de diseño "*Intercepting Filter*"[Alur, 2001], *Producer-Consumer* [Grand, 1998] y "*Strategy*" [Gamma, 1995], los cuales se reflejan en diagrama de clases de la Figura 6.4.

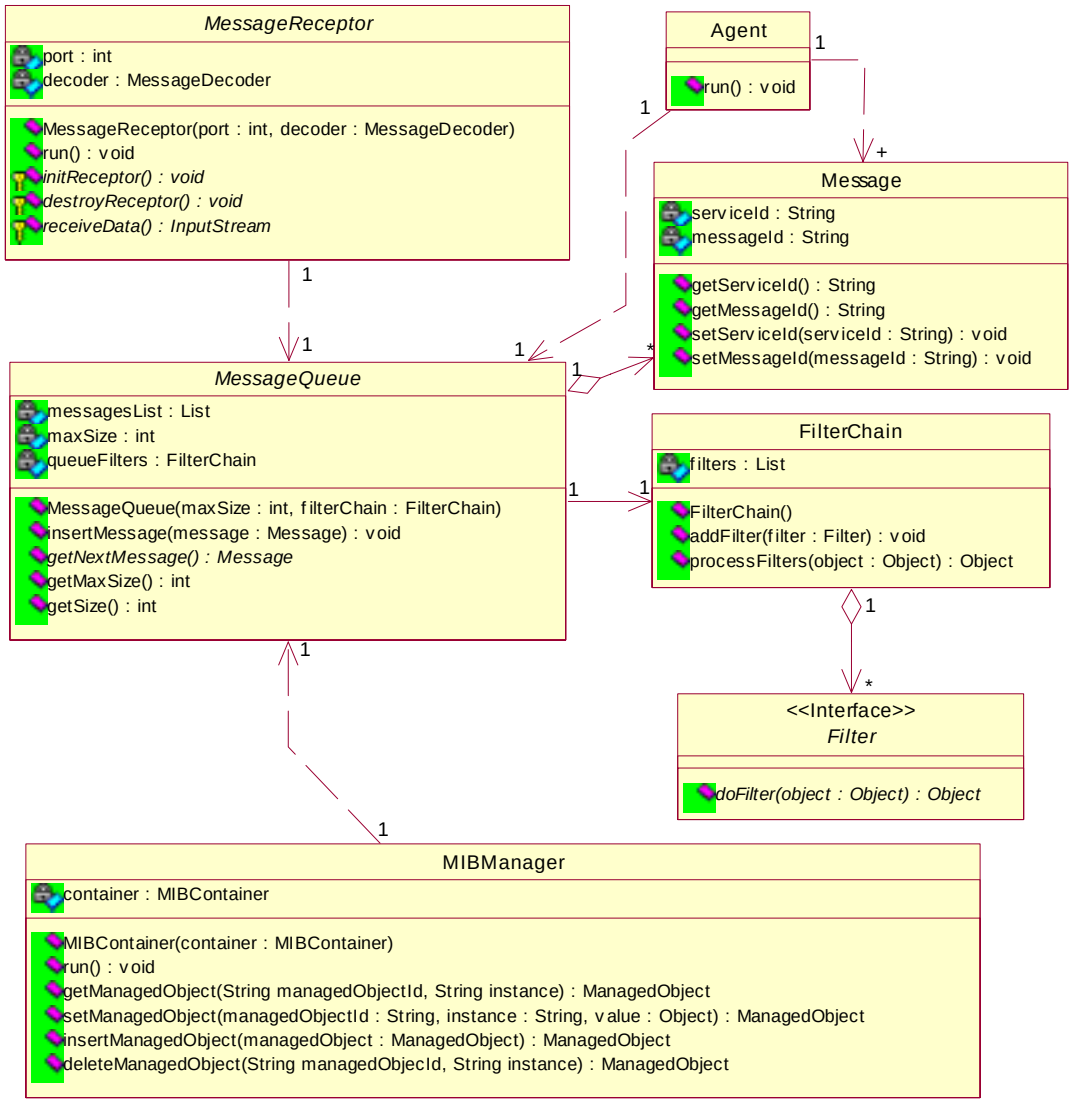


Figura 6.4 Modelo Estático de la Solución de Diseño a los Conceptos "*Lista de Espera de Solicitudes*", "*Filtro de Envío de Eventos*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Intercepting Filter***. El método *insertMessage(…)* que se encuentra en la clase *MessageQueue* aplica la cadena de filtros definida en una instancia de la clase *FilterChain*, la cual se encarga de aplicar los filtros de forma ordenada y secuencial. Los filtros pueden ser de encriptación, descompactación, de seguridad, de discriminación de envío de eventos, etc. . En caso de que un mensaje sea descartado en la aplicación de uno de los filtros, éste no es insertado en la lista de espera, en caso contrario, significa que el mensaje cumple con los discriminantes de los filtros, por lo cual el mensaje es insertado en la lista de espera.

- ***Producer-Consumer***. El patrón de diseño *Producer-Consumer* es implementado a través de la interacción entre la instancia de la clase *Agent* y la instancia de *MessageQueue*, la cual le permite consumir a la instancia *Agent* (método *run(…)*) los mensajes producidos por la instancia MesageReceptor y MIB.

- ***Strategy***. Este patrón de diseño es reflejado en el método *getNextMessage(…)*, ya que el Agente delega a la instancia de *MessageQueue* la responsabilidad de la obtención del siguiente mensaje a analizar. El método *getNextMessage(…)* es abstracto, por lo cual el algoritmo para la obtención del mensaje debe ser implementado por el desarrollador del Agente.

6.5.2.6 Descripción de Clases

La descripción de las clases *MessageQueue, FilterChain, Filter* se encuentra definida el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.2 Descripción de Clases del Componente Agente*".

6.5.2.7 Lógica de Control del Framework

Parte de la lógica de control del framework se encuentra reflejada en el método *insertMessage(…)* de la clase *MessageQueue* y en el método *processFilters(…)* ya que a través de ellos se aplica la cadena de filtros definida para la cola de mensajes del Agente, lo cual permite discriminar los mensajes que el Agente debe atender. Por lo tanto estos métodos no deben ser redefinidos a menos que se requiera modificar la lógica de almacenamiento y filtrado de mensajes.

6.5.2.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *MessageQueue*, y *Filter*.

MessageQueue. Clase encargada de filtrar y almacenar mensajes ya sea provenientes del Administrador, o provenientes del AVH los cuales reportan cambios de estado en los Recursos. Los métodos que deben ser concretizados son los siguientes:

- ***getNextMessage(…)***. Permite obtener el siguiente mensaje almacenado en la cola. Este método es abstracto debido a que no se sabe si debe existir un mecanismo diferente a FIFO para obtener el siguiente mensaje en la cola, ya que puede existir un mecanismo que atienda primero a aquellos mensajes que tienen prioridad, etc.

Filter. Interfaz encargada de encapsular la lógica de negocio de filtros que se deben aplicar sobre los mensajes. Los métodos que deben ser concretizados son los siguientes:

- ***doFilter(…)***. Contiene la lógica de discriminación que debe aplicarse a los mensajes. El comportamiento esperado es que si el mensaje no cumple con los requisitos de discriminación, éste sea desechado retornando null en vez del objeto *Message*, esto para que la cola de mensajes no lo tomen en cuenta.

6.5.2.9 Modelo Dinámico

La Figura 6.5 presenta el modelo dinámico entre instancias de las clases *MessageQueue, FilterChain* y *Filter*.

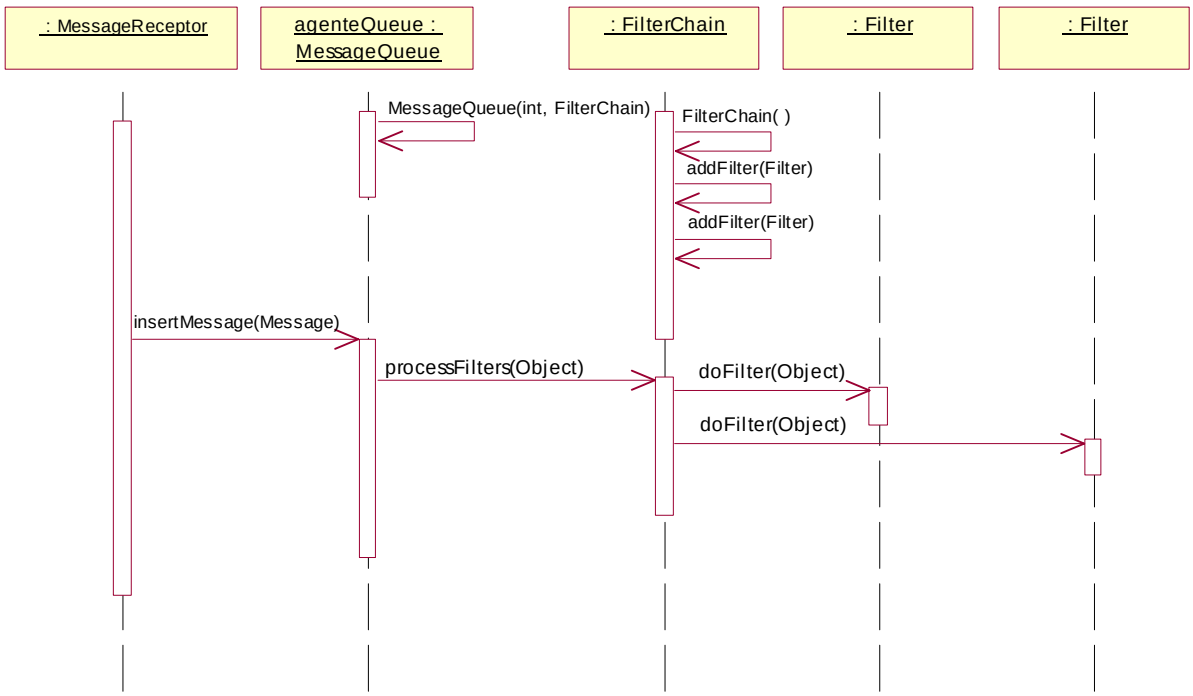


Figura 6.5 Modelo Dinámico de la Solución de Diseño a los Conceptos "*Lista de Espera de Solicitudes*" y "*Filtro de Envío de Eventos*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. Se inicia el mecanismo de filtrado de mensajes definiendo los filtros que deben aplicarse. Esto se realiza en el momento en que se configura la aplicación, utilizando el método *addFilter(…)* de *FilterChain*. En este caso se definen dos filtros.
2. Cuando la instancia de *MessageReceptor* recibe el flujo de datos y lo convierte en un objeto *Message*, inserta el mensaje en la cola del Agente mandando llamar al método *insertMessage(…)* de *agentQueue.*
3. *agentQueue* a su vez manda llamar el método *processFilters(…)* de *FilterChain* para que le sean aplicados los filtros al mensaje.
4. Se aplican los filtros a través de la llamada *doFilter(…)* de las instancias de *Filter*.
5. Por último, si los filtros aplicados indican que el mensaje cumple con los discriminantes, se inserta el mensaje en la cola, en caso contrario se descarta.

6.5.3 Agente, Administrador de Procesos, Proceso y Servicio

El diseño de la solución de los conceptos *Agente*, *Administrador de Procesos*, *Proceso*, y *Servicio* se presenta de manera conjunta debido a que éstos están muy relacionados entre sí, ya que en ellos se concentra la funcionalidad principal del Agente, por lo cual es más comprensible el presentar el diseño de solución de manera conjunta a presentarla de manera individual.

6.5.3.1 Responsabilidad

- El *Agente* interpreta y ejecuta comandos enviados por el Administrador a través de mensajes, además permite enviar reportes de cambio de estado en los Objetos Administrados al Administrador.
- El *Administrador de Procesos* permite obtener de un pool un proceso que permite la ejecución de un servicio.
- El *Proceso* ejecuta el servicio solicitado por el administrador.
- El *Servicio* contiene la lógica de negocio que debe utilizarse para atender la solicitud del Administrador y enviarle los resultados correspondientes.

6.5.3.2 Puntos Comunes

Los puntos comunes del concepto *Agente* son:

- Servicio que interpreta y ejecuta comandos enviados por el Administrador de una manera concurrente.

Los puntos comunes del concepto *Administrador de Procesos* son:

- Servicio común en los Agentes, el cual es independiente del modelo de información implementado. Este servicio administra, crea, suspende y elimina los procesos utilizados para la ejecución de los servicios solicitados por el Administrador.

Los puntos comunes del concepto *Proceso* son:

- Elemento utilizado para la ejecución de un servicio, el cual es independiente del modelo de información implementado en el Agente.

Los puntos comunes del concepto *Servicio* son:

- Ejecución de un comando solicitado por el administrador.

6.5.3.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *Agente* son:

- Protocolo de Administración Red utilizado para la recepción y el envío de mensajes.
- Variedad de servicios a ejecutar.
- Servicios que deben ser ejecutados antes de interpretar y ejecutar un mensaje (Este punto de variabilidad fue delegado al concepto "*Lista de Espera de Solicitudes*").

Los puntos de variabilidad del concepto *Administrador de Procesos* son:

- Generación y obtención de procesos según la demanda.
- La suspensión, recuperación y eliminación de procesos según corresponda.

Para el concepto *Proceso* no existen puntos de variabilidad.

Los puntos de variabilidad del concepto *Servicio* son:

- Diversidad de servicios y lógica que debe ser implementada para la ejecución de comandos enviados por el Administrador.
- Modo confirmado o no confirmado, es decir, si el servicio debe retornar o no los resultados al Administrador.
- Tipo de ejecución permitida, es decir si el servicio es atómico (todo o nada) o mejor esfuerzo (lo que sea posible ejecutar).

6.5.3.4 Solución

Dado que el Agente se encarga de obtener el mensaje siguiente a interpretar y ejecutar, es necesario que el comando a ejecutarse no bloquee la interpretación y ejecución de otros comandos, es por ello que es necesario utilizar al Administrador de Procesos, el cual obtiene un proceso que ejecuta el servicio y permite al Agente atender a los demás mensajes lo cual permite la concurrencia en el Agente. Esta lógica representa al "*Punto Común*" del Agente. El protocolo de Administración de Red utilizado por el Agente y la variedad de servicios que el Agente puede ejecutar quedan representados por el identificador del servicio encapsulado en el siguiente mensaje a interpretar, el Agente en este caso obtiene de la configuración del framework el servicio que le corresponde a la solicitud dado el identificador del servicio solicitado. Esta lógica representa a los "*Puntos de Variabilidad*" del Agente

El Administrador de Procesos le permite al Agente obtener un proceso y encapsular en el Proceso el servicio a ejecutar, la lógica de la administración de los procesos y cómo éstos entran y salen del pool representa el "*Punto Común*" del Administrador de Procesos. No obstante la forma en que se obtiene el proceso y se inicia el pool es abstracta, y representan los "*Puntos de Variabilidad*" del Administrador de Procesos.

El Proceso ejecuta el servicio el cual contiene la lógica de negocio necesaria para realizar las actividades solicitadas por el Administrador, en este caso el proceso puede ser reutilizado asignando el servicio a ejecutar, ejecutar el servicio, y después de su ejecución liberarlo para que se encuentre disponible para la ejecución de otro servicio. Esto representa el "Punto Común" de éste concepto.

Por último el Servicio permite ejecutar un comando, lo cual representa el "*Punto Común*" de éste concepto, no obstante la lógica de negocio y actividades utilizadas para la ejecución de un servicio en particular y si éste debe enviarle al Administrador el mensaje de respuesta, representan los "*Puntos de Variabilidad*" de éste concepto.

6.5.3.5 Modelo Estático

El modelo estático de los conceptos *Agente, Administrador de Procesos, Proceso,* y *Servicio* consiste en la aplicación de los patrones de diseño *Template Method* [Gamma, 1995], *Producer-Consumer* [Grand, 1998], *Dynamic Linkage* [Grand, 1998], *ObjectPool* [Grand, 1998], *Strategy* [Gamma, 1995], *Command* [Gamma, 1995] y *Singleton* [Gamma, 1995], los cuales se reflejan en diagrama de clases de la Figura 6.6.

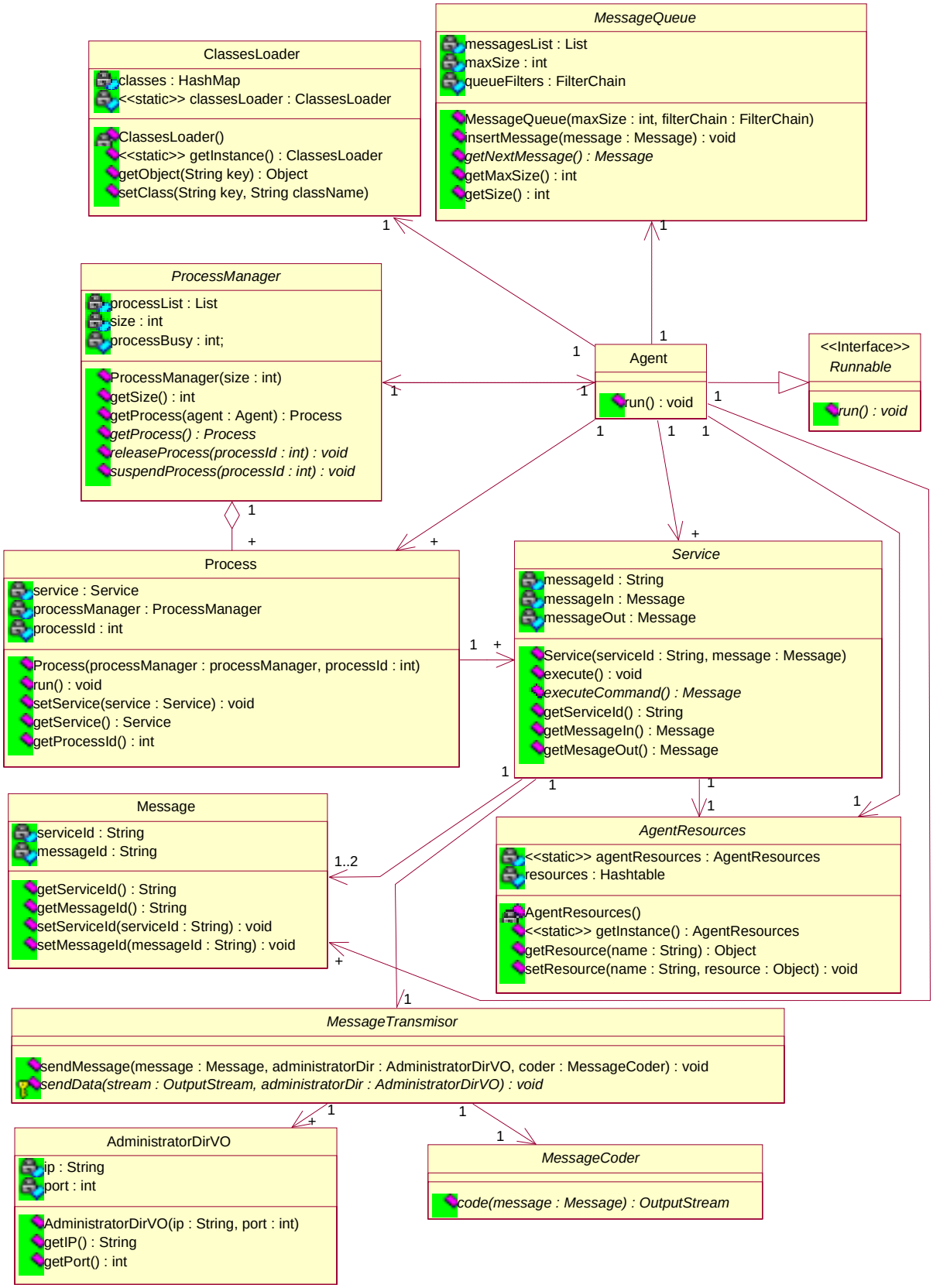


Figura 6.6 Modelo Estático de la Solución de Diseño a los Conceptos "*Agente*", "*Administrador de Procesos*", "*Proceso*" y "*Servicio*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Template Method***. El método plantilla se encuentra en la clase *Agent* en su método *run(…)*, el cual contiene la lógica suficiente para interpretar y ejecutar el servicio solicitado. Este patrón de diseño es implementado también en instancias de la clase *Process*, ya que en el método *run(…)* se encuentra plasmado el método plantilla que permite la ejecución de instancias *Service* y a su vez se comunica con la instancia de la clase *ProcessManager* cuando se termina la ejecución del servicio, para que se ponga disponible en el pool de procesos.

- ***Producer-Consumer***. El patrón de diseño *Producer-Consumer* es implementado a través de la interacción entre la instancia de la clase *Agent* y la instancia de *MessageQueue*, la cual produce los mensajes (*método getNext(…)*) que la instancia *Agent* consume en su método principal *run(…)*.

- ***Dynamic Linkage***. Este patrón de diseño es implementado a través de la única instancia de *ClassesLoader*, el cual dado un identificador permite obtener objetos de manera dinámica sin saber a que clase pertenece la instancia solicitada. Este patrón de diseño es muy útil ya que le permite a la instancia de la clase *Agent* obtener el servicio a ejecutar dado el identificador encapsulado en objeto *Message* (*serviceId*).

- ***Object Pool***. Este patrón de diseño es implementado a través de la instancia *ProcessManager*, el cual mantiene un pool de instancias *Process*, lo que permite a la instancia de la clase *Agent* ejecutar concurrentemente los servicios solicitados por el Administrador.

- ***Strategy***. El método plantilla de la instancia de la clase *Agent* delega a la instancia de la clase *ProcessManager* la responsabilidad de la obtención de un objeto *Process* para la ejecución de un servicio (Método *getProcess(…)*). También el método plantilla de la clase *Process* delega a la instancia correspondiente de la clase *Service* la lógica de negocio y las actividades a realizar para proporcionar el servicio solicitado (Método *execute(…)*). Por último en el método *sendMessage(…)* le delega a la clase *MessageCoder* la responsabilidad de codificar el mensaje (Método *code(…)*).

- ***Command***. Este patrón de diseño está reflejado en el método *executeCommand()* de la clase *Service* ya que éste método abstrae las actividades que se deben realizar dado la solicitud de ejecución de un comando por parte del Administrador. Este método en particular es importante en el diseño del framework, ya que permite al mismo ser flexible en cuanto a la adaptación de nuevos comandos dadas las actualizaciones del Protocolo de Administración de Red utilizado.

- ***Singleton***. Este patrón de diseño se encuentra reflejado en las clases *AgentResources* y *ClassesLoader* ya que sólo se puede obtener una sola instancia de estas clases, lo cual impide instanciarlas directamente, y sólo se puede obtener una sola instancia de éstas a través del método *getInstance()*. Este patrón se utilizó ya que es necesario que no se generen varias instancias de estas clases ya que una contiene los Recursos disponibles en el Agente los cuales deben ser únicos y la otra permite el enlace dinámico.

6.5.3.6 Descripción de Clases

La descripción de las clases *Agent, AgentResources, ClassesLoader, ProcessManager, Process, Service, MessageTransmisor, MessageCoder MessageCoder* y *AdministratorDirVO* se encuentra en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.2 Descripción de Clases del Componente Agente*".

6.5.3.7 Lógica de Control del Framework

Gran parte de la lógica de control del framework se encuentra reflejada en el método *run(…)* de la clase *Agent* ya que tiene la responsabilidad de obtener los mensajes almacenados en la cola, y atenderlos, tomando decisiones como el comando que debe ejecutarse dado el mensaje, obtener un proceso a través de *ProcessManager* y asignarle el servicio correspondiente para que la solicitud pueda ser atendida, realizando todo esto de manera concurrente.

El método *run(…)* de la clase *Process* contiene lógica de control respecto a la secuencia de actividades que deben seguirse para la ejecución de un servicio. una muy importante es la de liberar el proceso una vez que el servicio se ejecutó, poniéndose disponible una vez maz en el *pool* de procesos de *ProcessManager*.

También es importante la colaboración del método *execute(…)* de la clase *Service* en el cual se define la lógica que debe seguirse en caso de que al ejecutar un servicio se tenga como respuesta un mensaje, enviando el mismo al Administrador.

Tanto el método *run(…)* de la clase *Agent*, el método *run(…)* de la clase *Process* como el método *execute(…)* de la clase *Service(…)* no deben ser redefinidos, a menos que se requiera modificar la lógica de atención y ejecución de servicios en el Agente.

6.5.3.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *ProcessManager*, *Service*, *MessageTransmisor* y *MessageCoder*.

ProcessManager. Clase encargada de mantener un *pool* de procesos disponibles para la ejecución de servicios. Los métodos que deben ser concretizados son los siguientes:

- ***getProcess(…)***. Permite obtener un proceso del *pool*. Este método es abstracto debido a que no se sabe con exactitud si va a existir un pool de procesos o éstos van a ser generados sobre demanda. El comportamiento esperado de este método es que cuando sea llamado por el Agente, le proporcione un objeto *Process* el cual ejecute un servicio, ya sea obteniéndolo del *pool* o generándolo de acuerdo a la demanda.

- ***releaseProcess(…)***. Libera un proceso y lo pone disponible para la ejecución de otro servicio. Este método es abstracto debido a que no se sabe si va a existir un *pool* de procesos o si éstos se dan sobre demanda. El comportamiento esperado es que dado el identificador del proceso, éste se ponga disponible para la ejecución de otro servicio, o en caso de que no exista un *pool*, se liberen los recursos de software utilizados por el objeto *Process*.
- ***suspendProcess(…)***. Suspende la ejecución de un proceso y lo pone disponible para la ejecución de otro servicio. Al igual que los métodos anteriores, este método es abstracto debido a que no se sabe si va a existir un *pool* de procesos o si éstos se dan sobre demanda. El comportamiento esperado es que dado el identificador del proceso, se cancele la ejecución del mismo y se ponga nuevamente disponible en el *pool* o se liberen los recursos utilizados por el objeto *Process*.

Service. Clase encargada de encapsular la lógica de negocio que debe seguirse al ejecutar un servicio determinado. Los métodos que deben ser concretizados son los siguientes:

- ***executeCommand(…)***. Contiene la lógica de negocio correspondiente a la ejecución de un comando de un Protocolo de Administración de Red en particular. Este método es abstracto debido a la diversidad de servicios que deben ser implementados para que un Agente soporte a un Protocolo de Administración de Red. El comportamiento esperado de este método es que dentro del mismo se defina la lógica de negocio que permita la ejecución de un comando, utilizando objetos auxiliares para obtener recursos de software los cuales le permitan cumplir con su objetivo. La forma en que se pueden obtener los recursos auxiliares es através de la única instancia de *AgentResources*, el cual por ejemplo permite obtener la referencia al *MIBManager* para poder realizar acciones sobre Objetos Administrados.

MessageTransmisor. Clase encargada de enviar mensajes por parte del Agente al Administrador. Los métodos que deben ser concretizados son los siguientes:

- ***sendData(…)***. Envía un mensaje al Administrador ya codificado en un flujo de datos. Este método es abstracto debido a la diversidad de mensajes que pueden ser enviados, ignorando la forma en que éstos pueden ser convertidos a un flujo de datos, además de no saber con certeza el tipo de protocolo utilizado para el envío de mensajes, el cual puede ser orientado o no a conexión. El comportamiento esperado de este método es que en él se defina la lógica necesaria para que independientemente de los tipos de mensajes que deban ser enviados, éste los convierta en un flujo de datos fácil de transmitir a través de la red.

MessageCoder. Clase encargada de codificar mensajes que se van a enviar al Administrador. El método que debe ser concretizado es el siguiente:

- ***code(…)***. Codifica un mensaje para convertirlo en un flujo de salida de datos. El comportamiento esperado es que este método lea el identificador de mensaje del objeto *Message* y elija el mecanismo de codificación necesario para la conversión a un flujo de salida, retornando el flujo a la instancia de *MessageTransmisor*.

6.5.3.9 Modelo Dinámico

La Figura 6.7(a) y Figura 6.7(b) presentan el modelo dinámico entre instancias de las clases *Agent, AgentResources, ClassesLoader, ProcessManager, Process, Service, MessageTransmisor, MessageCoder MessageCoder* y *AdministratorDirVO*.

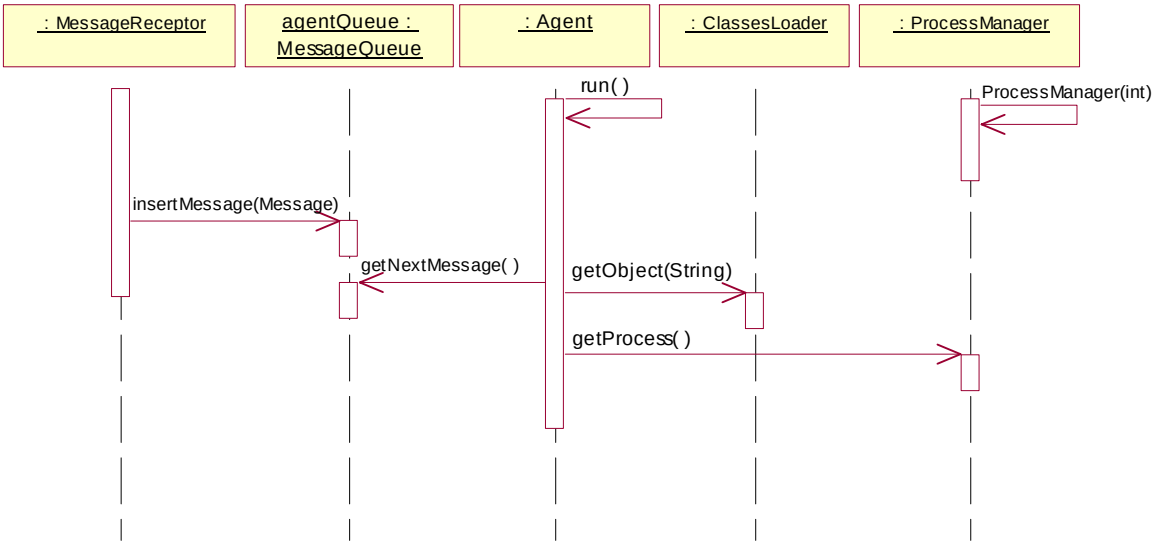


Figura 6.7(a) Modelo Dinámico de la Solución de Diseño a los Conceptos "*Agente*", "*Administrador de Procesos*", "*Proceso*" y "*Servicio*".

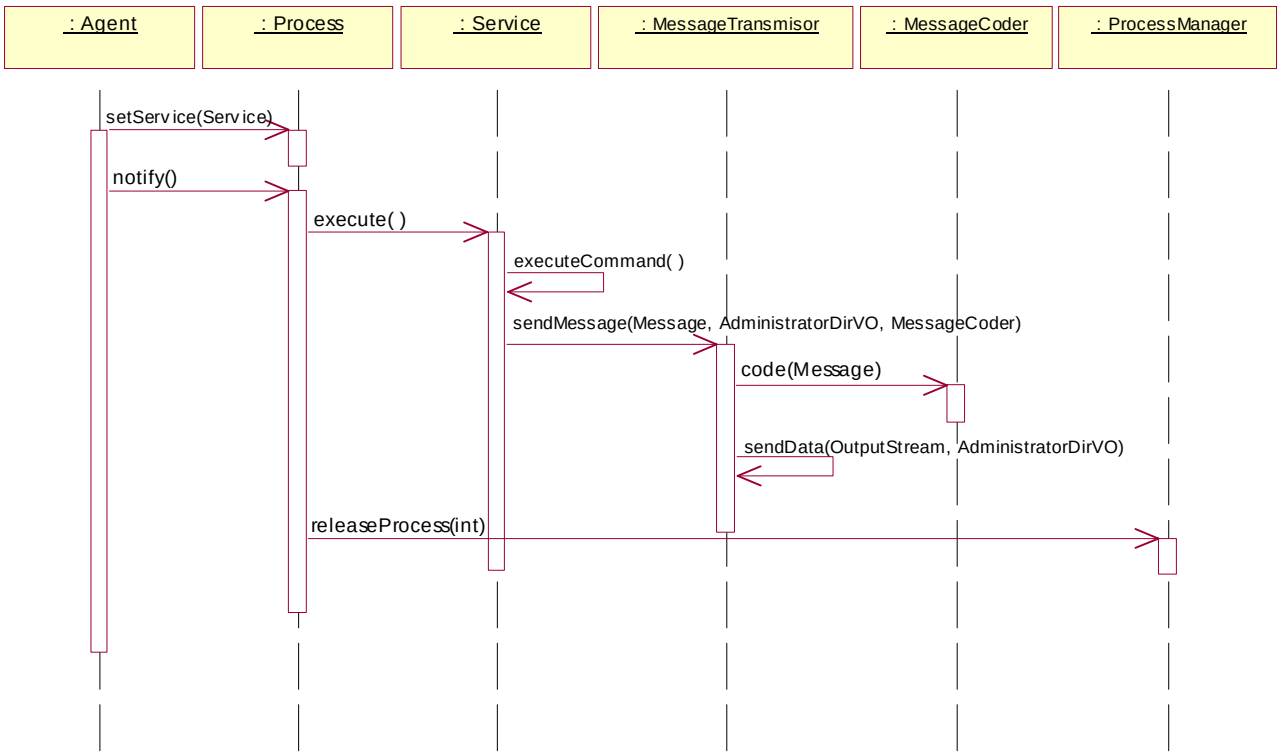


Figura 6.7(b) Modelo Dinámico de la Solución de Diseño a los Conceptos "*Agente*", "*Administrador de Procesos*", "*Proceso*" y "*Servicio*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. La instancia de *Agent* obtiene un mensaje a través de la llamada al método *getNextMessage(…)* de *agentMessage*.
2. *Agent* obtiene un objeto *Service* a través de la llamada al método *getObject(…)* de *ClassesLoader*, enviándole el identificador del servicio. El identificador del servicio le permite a *ClassesLoader* relacionarlo con el objeto *Class* correspondiente al servicio, creando así una instancia nueva.
3. Una vez que la instancia de *Agent* obtiene el objeto *Service*, obtiene un objeto *Process* del *pool* administrado por *ProcessManager*, a través de la llamada al método *getProcess(…)*.
4. *Agent* asigna el servicio al proceso obtenido, llamando al método *setService(…)* de la instancia de la clase *Process*.
5. *Agent* "despierta" al proceso, lo cual inicia la ejecución del servicio.
6. Se ejecuta el servicio a través del método *execute(…)* de *Service*, el cual a su vez realiza la llamada al método *executeCommand(…)*.
7. La instancia de *Service* verifica si existe un mensaje de salida, el cual fue producto de la ejecución del servicio, y en caso afirmativo intenta enviar el mensaje a través de la llamada al método *sendMessage(…)* de la clase *MessageTransmisor*.
8. Se codifica el mensaje y se convierte en un flujo de salida a través de la llamada al método *code(…)* de una instancia de la clase *MessageCoder*.
9. La instancia de *MessageTransmisor* envía el mensaje a través de la llamada al método *sendData(…)*.
10. Por último, el proceso es liberado y puesto disponible en el *pool* de procesos de *ProcessManager*, tomando un estado de suspensión.

6.6 Diseño del Componente "MIB"

El diseño de la solución para el componente *MIB* debe cumplir con los siguientes objetivos:

1. **Administración del Contenedor de Objetos Administrados**. Debe existir sólo un punto en donde se permita la ejecución de acciones sobre Objetos Administrados, que en este caso será el Administrador de la MIB.
2. **Contenedor de Objetos Administrados Abstracto**. El contenedor de la MIB deberá ser abstracto debido a que éste depende de la capacidad de almacenamiento del Elemento de Red, y los recursos de software que puedan ser utilizados para la contención de Objetos Administrados, como pueden ser archivos, bases de datos, memoria, etc.
3. **Manejo de Reportes de Cambio de Estado en los Recursos**. El framework deberá ser lo suficientemente flexible para manejar los cambios de estado en los Recursos del Elemento de Red los cuales hayan sido reportados por el Administrador Virtual de Hardware. Además deberá existir una entidad en la cual todos éstos reportes sean analizados uno por uno y en su caso, sean notificados al Agente para que éste decida si se deben enviar o no éstos reportes al Administrador.
4. **Servicios de Filtrado de Reportes de Cambio de Estado**. El filtrado en los reportes de cambio de estado en los Recursos contará con una serie de servicios quienes analicen los reportes antes de que el Administrador de la MIB los atienda. Estos servicios pueden ser servicios de Bitácora, de análisis de severidad de alarma, discriminación de reportes, etc.

Una vez determinados los objetivos específicos que el framework deberá cumplir con el diseño del componente "*MIB*", se presenta a continuación para cada concepto asociado con el componente, su responsabilidad, los puntos comunes y de variabilidad determinados en el AVCD y la solución a éstos. Cabe destacar que los conceptos presentados en esta sección se encuentran ordenados de acuerdo al orden en cómo éstos participan en el MCDOD dado el caso de uso del dominio "*Ejecución de un Servicio en el Agente*" (Véase el "*Capítulo 4. Análisis de Dominio Orientado a Objetos*" en su sección "*4.12. Modelo Conceptual Dinámico de Objetos de Dominio*").

6.6.1 MIB, Buscador de Objetos Administrados, Analizador de Permisos, Objeto Administrado

El diseño de la solución de los conceptos *MIB, Buscador de Objetos Administrados, Analizador de Permisos,* y *Objeto Administrado* se presenta de manera agrupada debido a que éstos conceptos son muy dependientes entre sí, por lo cual por cuestiones de claridad y comprensión del diseño, éste se presenta de manera conjunta.

6.6.1.1 Responsabilidad

- La *MIB* se encarga de proporcionar a los servicios del Agente la interfaz requerida para la aplicación de acciones sobre Objetos Administrados, se encarga de utilizar también un contenedor abstracto para la contención de éstos objetos. Además analiza los cambios de estado que el *AVH* y los aplica a los Objetos Administrados, y en su caso los envía al Agente para que analice la posibilidad de enviarlos al Administrador.
- El *Buscador de Objetos Administrados* permite obtener dado un identificador, el Objeto Administrado solicitado.
- El *Analizador de Permisos* tiene como responsabilidad el análisis de los permisos de lectura y escritura con los que cuentan los Objetos Administrados.
- El *Objeto Administrado* encapsula la información referente a Recursos del Elemento de Red, tanto de hardware como de software.

6.6.1.2 Puntos Comunes

Los puntos comunes del concepto *MIB* son:

- Contenedor común en la implementación de Agentes el cual se encarga de almacenar a los Objetos Administrados.
- El modelo utilizado para la contención de los Objetos Administrados es un modelo jerárquico.

Los puntos comunes del concepto *Buscador de Objetos Administrados* son:

- Servicio común en las MIBs, el cual permite obtener las referencias hacia Objetos Administrados dado un identificador de clase y un identificador de instancia.

Los puntos comunes del concepto *Analizador de Permisos* son:

- Servicio común en las MIBs ya que verifica los permisos de acción sobre un Objeto Administrado.

Los puntos comunes del concepto *Objeto Administrados* son:

- Elemento común en implementaciones de MIBs, el cual contiene información relevante acerca de un Recurso de Elemento de Red.
- Tiene un identificador único y jerárquico.

6.6.1.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *MIB* son:

- Diversidad de Objetos Administrados a contener dependiendo del Elemento de Red al que pertenece.
- Diversos esquemas de contención y almacenamiento de Objetos Administrados, como lo pueden ser Bases de Datos, Archivos y la Memoria RAM.

Los puntos de variabilidad del concepto *Buscador de Objetos Administrados* son:

- Tipos de identificadores de búsqueda, los cuales dependen del modelo de información utilizado.
- Información adicional que puede existir para la búsqueda de un Objeto Administrado.

Los puntos de variabilidad del concepto *Analizador de Permisos* son:

- Diversidad de permisos que pueden ser analizados, como lo son lectura y escritura.

Los puntos de variabilidad del concepto *Objeto Administrados* son:

- Diversidad de clases de Objetos Administrados a la que puede pertenecer.
- Representación del Objeto Administrado, dependiendo del modelo de información utilizado.

6.6.1.4 Solución

La MIB debe contar con un punto de entrada, el cual permita tener contacto directo con el contenedor de los Objetos Administrados, al cual se denomina el Administrador de la MIB. El contenedor debe ser abstracto, de tal forma que sea adaptable al tipo de esquema de almacenamiento a utilizar. Este contenedor debe proporcionar soporte para las operaciones básicas como lo son el obtener, insertar y eliminar Objetos Administrados.

No obstante que el Administrador de MIB es un punto de entrada para realizar una acción en Objetos Administrados, también deberá atender las solicitudes de cambio de estado en los Recursos, es decir deberá contar con una cola de espera de reportes, los cuales sean atendidos uno a uno.

El Buscador de Objetos Administrados depende del tipo de información utilizada para la búsqueda de Objetos Administrados, por lo cual éste se deberá encontrar en clases concretas que implementen el contenedor de MIB.

Los "*Puntos de Variabilidad*" del concepto MIB y Buscador de Objetos Administrados se resuelven implementando la interfaz del contenedor de Objetos Administrados, ya que en ella recae la responsabilidad de implementar el esquema de almacenamiento a utilizar, encargándose también de la búsqueda de Objetos Administrados.

El Analizador de Permisos se encuentra dentro del Administrador de MIB, ya que al intentar realizar una acción sobre un Objeto Administrado, se verifica que éste cuente con los permisos de lectura o escritura necesarios para permitir tal acción.

Para implementar Objetos Administrados, siempre se deberá extender de la clase base que los representa, ya que en ella se encuentra información común la cual es manejada en ambos modelos de información (OSI e Internet). En este caso, la clase base representa los "*Puntos Comunes*" de este concepto, ya que en ella se encuentra la información común de este elemento, mientras que los "*Puntos de Variabilidad*" se resuelven extendiendo de esta clase base, ya que existe una diversidad importante de clases de Objetos Administrados.

6.6.1.5 Modelo Estático

El modelo estático de los conceptos *MIB, Buscador de Objetos Administrados, Analizador de Permisos,* y *Objeto Administrado* consiste en la aplicación de los patrones de diseño "*Access Proxy*" [Gamma, 1995], *Producer-Consumer* [Grand, 1998] y *Single Threaded Execution* [Grand, 1998] y *Adapter* [Gamma, 1995] los cuales se reflejan en diagrama de clases de la Figura 6.8.

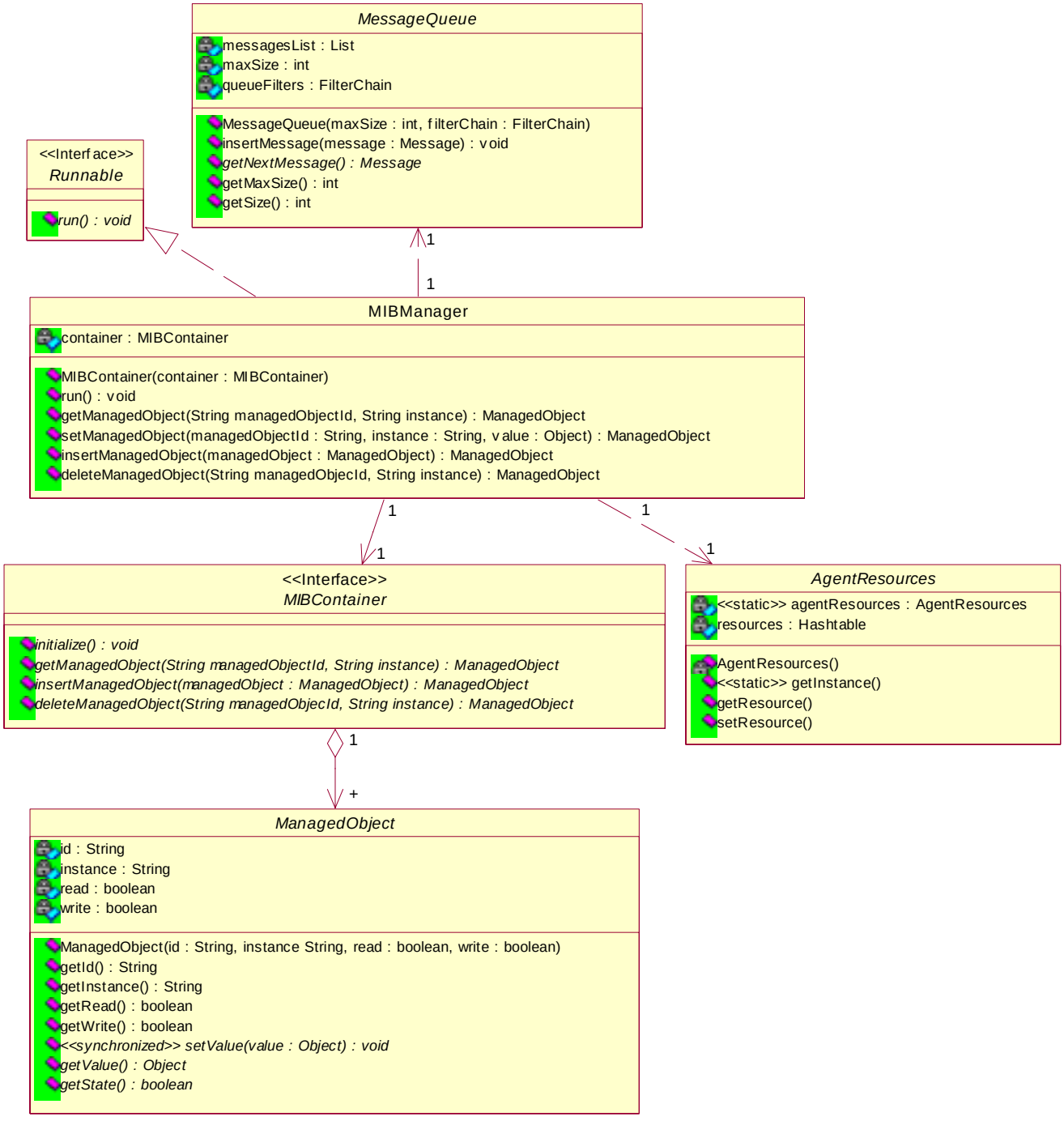


Figura 6.8 Modelo Estático de la Solución de Diseño a los Conceptos "*MIB*", "*Buscador de Objetos Administrados*", "*Analizador de Permisos*" y "*Objeto Administrado*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Access Proxy***. Este patrón de diseño es implementado a través de los métodos *getManagedObject(…), setManagedObject(…), insertManagedObject(…)* y *deleteManagedObject(…),* ya que permiten o niegan el acceso a los métodos del contenedor *MIBContainer*, y es solo a través de éstos métodos que se puede acceder a la funcionalidad del contenedor de Objetos Administrados.

- ***Producer-Consumer***. El patrón de diseño *Producer-Consumer* es implementado a través de la interacción entre la instancia de la clase *MIBManager* y la instancia de *MessageQueue,* la cual produce los mensajes (*método getNext()*) que la instancia *MIBManager* consume en su método principal *run(),* a través de éste mecanismo es posible que se atiendan los reportes de cambio de estado enviados por el Administrador Virtual de Hardware.

- ***Single Threaded Execution***. Este patrón de diseño está reflejado en el método *setValue(…)* de la clase *ManagedObject* ya que éste método es sincronizado y no es ejecutado concurrentemente por Threads, si no que sólo un Thread a la vez puede ejecutarlo. Esto permite que se pueda modificar el estado de un Objeto Administrado de manera determinante y síncrona.

- ***Adapter***. Este patrón de diseño es implementado en el método *setValue(…), getValue(…)* y *getState(…)* de la clase *ManagedObject*, ya que de manera abstracta este método deberá ser implementado por subclases de Objetos Administrados, los cuales tengan la lógica necesaria para adaptar estos métodos de acuerdo a sus necesidades y formatos de información que puedan recibir para modificar el estado del Objeto Administrado, obtener su valor, y obtener el estado actual del mismo, el cual permita verificar si es necesario reportar un cambio de estado crítico.

6.6.1.6 Descripción de Clases

La descripción de las clases *MIBManager, MIBContainer* y *ManagedObject* se encuentra en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.3 Descripción de Clases del Componente MIB*".

6.6.1.7 Lógica de Control del Framework

Parte de la lógica de control del framework se encuentra reflejada en el método *run(…)* de la clase *MIBManager* ya que tiene la responsabilidad de obtener los reportes de cambio de estado en los Recursos almacenados en la cola, y atenderlos, obteniendo el Objeto Administrado sobre el cual se debe reflejar el cambio de estado, y en caso de que ésta acción rebase los límites frontera del mismo, se genera un mensaje de cambio de estado el cual es insertado en la cola de mensajes del Agente. Este método por tanto no debe ser redefinido, a menos que se requiera modificar la lógica de atención a reportes de cambio de estado en Recursos administrador por la AVH.

6.6.1.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *MIBContainer* y *ManagedObject*.

MIBContainer. Interfaz que define los métodos que deben ser concretizados para la implementación del contenedor de Objetos Administrados utilizado por la MIB. Los métodos que deben ser concretizados son los siguientes:

- ***initialize(…)***. Inicializa el contenedor de la MIB. Este método es abstracto debido a que no se sabe con exactitud que actividades se deben realizar en el momento de poner disponible al contenedor de la MIB. El comportamiento esperado de este método es que realice las actividades previas a la disponibilidad del contenedor, inicializando por ejemplo conexiones a Bases de Datos donde se encuentran almacenados los Objetos Administrados o en su defecto, generar instancias de Objetos Administrados y contenerlos en una estructura de datos que permita su fácil obtención cuando así sea requerido.
- ***getManagedObject(…)***. Obtiene un Objeto Administrado dado un identificador de clase y un identificador de instancia. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder obtener Objetos Administrados. El comportamiento esperado es que obtenga la instancia del Objeto Administrado solicitado de forma que utilice el mecanismo de contención definido.
- ***insertManagedObject(…)***. Inserta en el contenedor un Objeto Administrado. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder insertar Objetos Administrados. El comportamiento esperado es que inserte la instancia del Objeto Administrado en el mecanismo de contención definido.
- ***deleteManagedObject(…)***. Elimina del contenedor un Objeto Administrado. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder eliminar Objetos Administrados. El comportamiento esperado es que elimine la instancia del Objeto Administrado en el mecanismo de contención definido.

ManagedObject. Clase que encapsula la información referente a un Objeto Administrado. Los métodos que deben ser concretizados son:

- ***setValue(…)***. Cambia el valor o estado del Objeto Administrado. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder modificar al Objeto Administrado. El comportamiento esperado es que se actualice el estado del Objeto Administrado y en caso de que los límites frontera de los valores del Objeto Administrado sean rebasados o en caso de que no sea adecuado el cambio de estado, este método debe retornar *false*, en caso contrario se retorna *true*.
- ***getValue(…)***. Obtiene el valor o estado del Objeto Administrado. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder obtener el valor de un Objeto Administrado. El comportamiento esperado es que se obtenga el estado del Objeto Administrado utilizando el mecanismo de contención definido.
- ***getState(…)***. Verifica el estado actual del Objeto Administrado. Este método es abstracto debido a que no se sabe con exactitud el mecanismo de contención utilizado ya sea memoria, archivos o Bases de Datos, para poder obtener el estado de un Objeto Administrado. El comportamiento esperado es que verifique que los valores frontera del Objeto Administrado o su estado sean los adecuados, en caso de que no sea así se retorna *false*, en caso contrario se retorna *true*.

6.6.1.9 Modelo Dinámico

La Figura 6.9 presenta el modelo dinámico entre instancias de las clases *Service, MIBManager, MIBContainer* y *ManagedObject, ReportMessage* y *MessageTransmisor*.

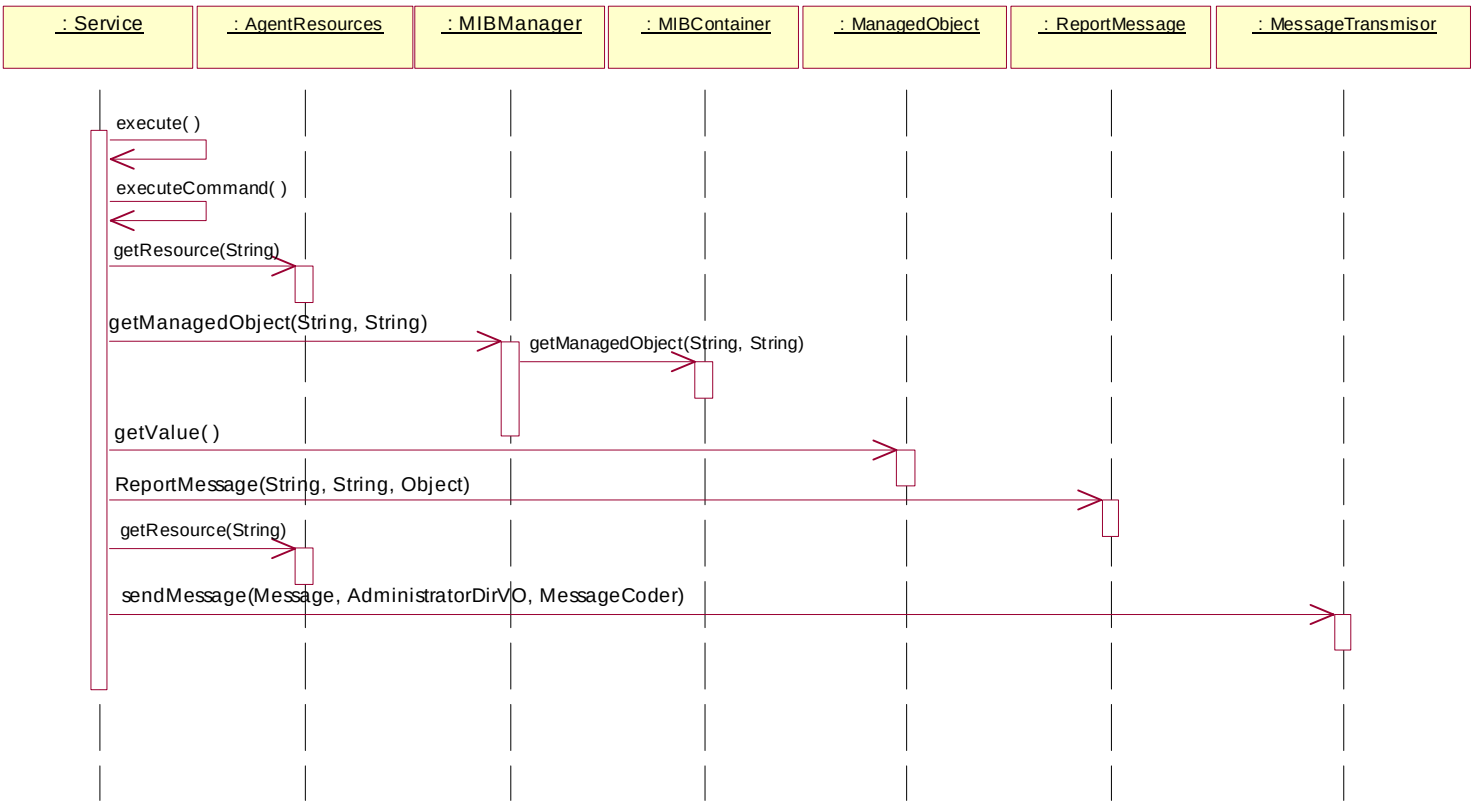


Figura 6.9 Modelo Dinámico de la Solución de Diseño a los Conceptos "*MIB*", "*Buscador de Objetos Administrados*", "*Analizador de Permisos*" y "*Objeto Administrado*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. La instancia de *Service* intenta obtener el valor de un Objeto Administrado, por lo cual obtiene la instancia de *MIBManager* a través de la llamada al método *getResource(…)* de *AgentResource*.
2. *Service* obtiene el Objeto Administrado a través de la llamada al método *getManagedObject(…)* de *MIBManager*, el cual a su vez realiza la llamada al método *getManagedObject(…)* de *MIBContainer*.
3. La instancia de *Service* obtiene el valor del *ManagedObject* a través de la llamada al método *getValue(…)*.
4. Al verificar que el estado del objeto no es el adecuado, *Service* genera una instancia de *ReportMessage*, esto con el fin de reportar el estado del Objeto Administrado.
5. *Service* envía el reporte de cambio de estado al Administrador a través de la llamada al método *sendMessage(…)*.

6.6.2 Filtro de Procesamiento de Bitácora, Servicio Analizador de Severidad de Alarma, Servicio de Bitácora

El diseño de la solución de los conceptos *Filtro de Procesamiento de Bitácora, Servicio Analizador de Severidad de Alarma* y *Servicio de Bitácora* se presenta de manera agrupada debido a que éstos conceptos representan filtros que deben ser aplicados a reportes de cambio de estado en los Recursos, por lo cual es mejor agruparlos en el diseño a que se diseñen de manera separada.

6.6.2.1 Responsabilidad

- El *Filtro de Procesamiento de Bitácora* tiene como responsabilidad el verificar si un reporte de cambio de estado en un Recurso debe ser almacenado en Bitácora.
- El *Servicio Analizador de Severidad de Alarma* permite obtener el grado de severidad con que cuenta el reporte de cambio de estado de un Recurso.
- El *Servicio de Bitácora* tiene como responsabilidad el almacenar los reportes de cambio de estado en los Recursos del Elemento de Red.

6.6.2.2 Puntos Comunes

- El *Filtro de Procesamiento de Bitácora,* el *Servicio Analizador de Severidad de Alarma* y el *Servicio de Bitácora* no cuentan con puntos comunes entre los modelos de información OSI e Internet debido a que estos conceptos se encuentran presentes en el modelo OSI pero no se encuentran definidos en el modelo de Internet.

6.6.2.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *Filtro de Procesamiento de Bitácora* son:

- Presente en el modelo OSI y no en el modelo de Internet.
- Diversidad de reportes de cambio de estado a analizar.

Los puntos de variabilidad del concepto *Servicio Analizador de Severidad de Alarma* son:

- Presente en el modelo OSI y no en el modelo de Internet.
- Diversidad de reportes de cambio de estado a analizar.

Los puntos de variabilidad del concepto *Servicio de Bitácora* son:

- Presente en el modelo OSI y no en el modelo de Internet.

6.6.2.4 Solución

Debido a que estos tres elementos pueden o no estar presentes en la implementación de la MIB, se opta por contemplarlos a todos como filtros al insertar un reporte de cambio de estado en un Recurso, ya que la cadena de filtros es lo suficientemente flexible como para añadir o eliminar servicios de análisis de mensajes o en este caso de reportes.

Cabe destacar que estos filtros deben ser configurados al momento de la inicialización del framework o en su caso se deben omitir si es que no son aplicables.

6.6.2.5 Modelo Estático

El modelo estático de los conceptos *Filtro de Procesamiento de Bitácora*, *Servicio Analizador de Severidad de Alarma* y *Servicio de Bitácora* consiste en la aplicación del patrón de diseño "*Intercepting Filter*" [Alur, 2001] el cual se refleja en diagrama de clases de la Figura 6.10.

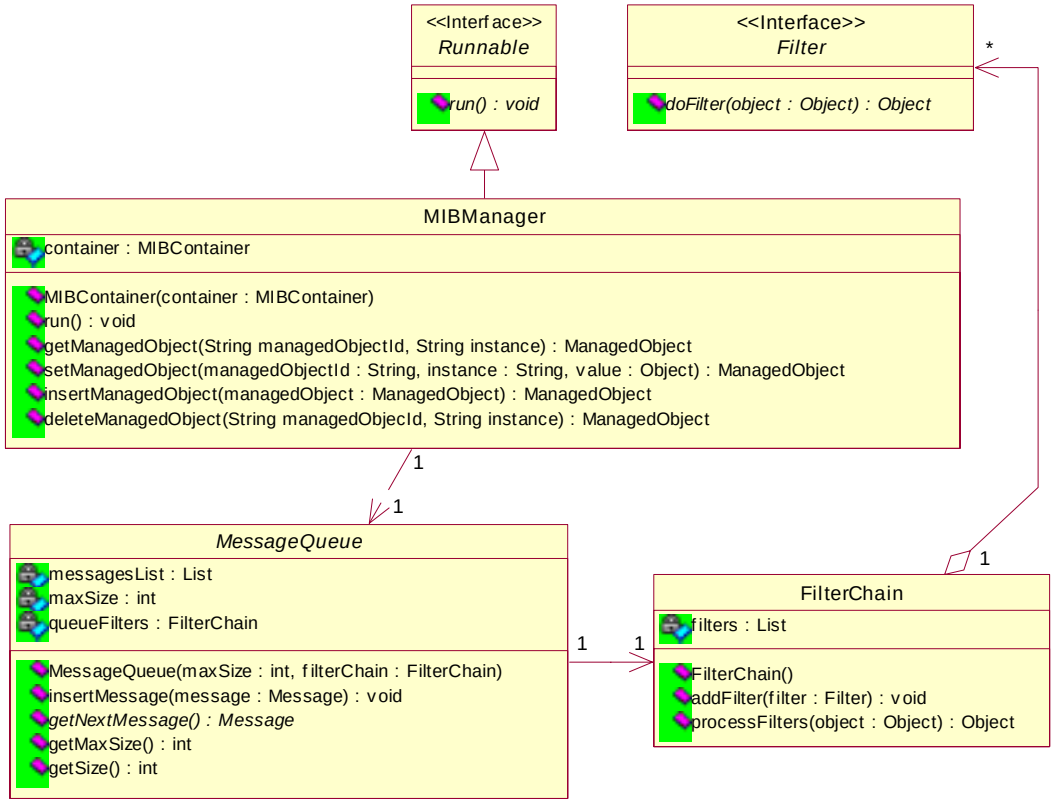


Figura 6.10 Modelo Estático de la Solución de Diseño a los Conceptos "*Filtro de Procesamiento de Bitácora*", "*Servicio Analizador de Severidad de Alarma*" y "*Servicio de Bitácora*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Intercepting Filter***. El método *insertMessage(…)* que se encuentra en la clase *MessageQueue* aplica la cadena de filtros definida en una instancia de la clase *FilterChain*, la cual se encarga de aplicar los filtros de forma ordenada y secuencial. Los filtros pueden ser análisis de severidad de alarma, de procesamiento de bitácora y de servicio de bitácora. En este caso los filtros no son discriminantes, sino que permiten la ejecución de servicios diversos que pueden o no estar presentes en la implementación de la MIB.

6.6.2.6 Descripción de Clases

La descripción de las clases *MIBManager, MessageQueue, FilterChain* y *Filter* se encuentra definida en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.3 Descripción de Clases del Componente MIB*".

6.6.2.7 Lógica de Control del Framework

Parte de la lógica de control del framework se encuentra reflejada en el método *insertMessage(…)* de la clase *MessageQueue* y en el método *processFilters(…)* ya que a través de ellos se aplica la cadena de filtros definida para la cola de mensajes del Administrador de la MIB, lo cual permite discriminar los mensajes que el Administrador de la MIB debe atender. Por lo tanto estos métodos no deben ser redefinidos a menos que se requiera modificar la lógica de almacenamiento y filtrado de mensajes.

6.6.2.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *MessageQueue*, y *Filter*.

MessageQueue. Clase encargada de filtrar y almacenar mensajes provenientes del AVH los cuales reportan cambios de estado en los Recursos. Los métodos que deben ser concretizados son los siguientes:

- ***getNextMessage(…)***. Permite obtener el siguiente mensaje almacenado en la cola. Este método es abstracto debido a que no se sabe si debe existir un mecanismo diferente a FIFO para obtener el siguiente mensaje en la cola, ya que puede existir un mecanismo que atienda primero a aquellos mensajes que tienen prioridad, etc.

Filter. Interfaz encargada de encapsular la lógica de negocio de filtros que se deben aplicar sobre los mensajes. Los métodos que deben ser concretizados son los siguientes:

- ***doFilter(…)***. Contiene la lógica de discriminación que debe aplicarse a los mensajes. El comportamiento esperado es que si el mensaje no cumple con los requisitos de discriminación, éste sea desechado retornando null en vez del objeto *Message*, esto para que la cola de mensajes no lo tomen en cuenta.

6.6.2.9 Modelo Dinámico

La Figura 6.11 presenta el modelo dinámico entre instancias de las clases *MIBManager, MessageQueue, FilterChain* y *Filter*.

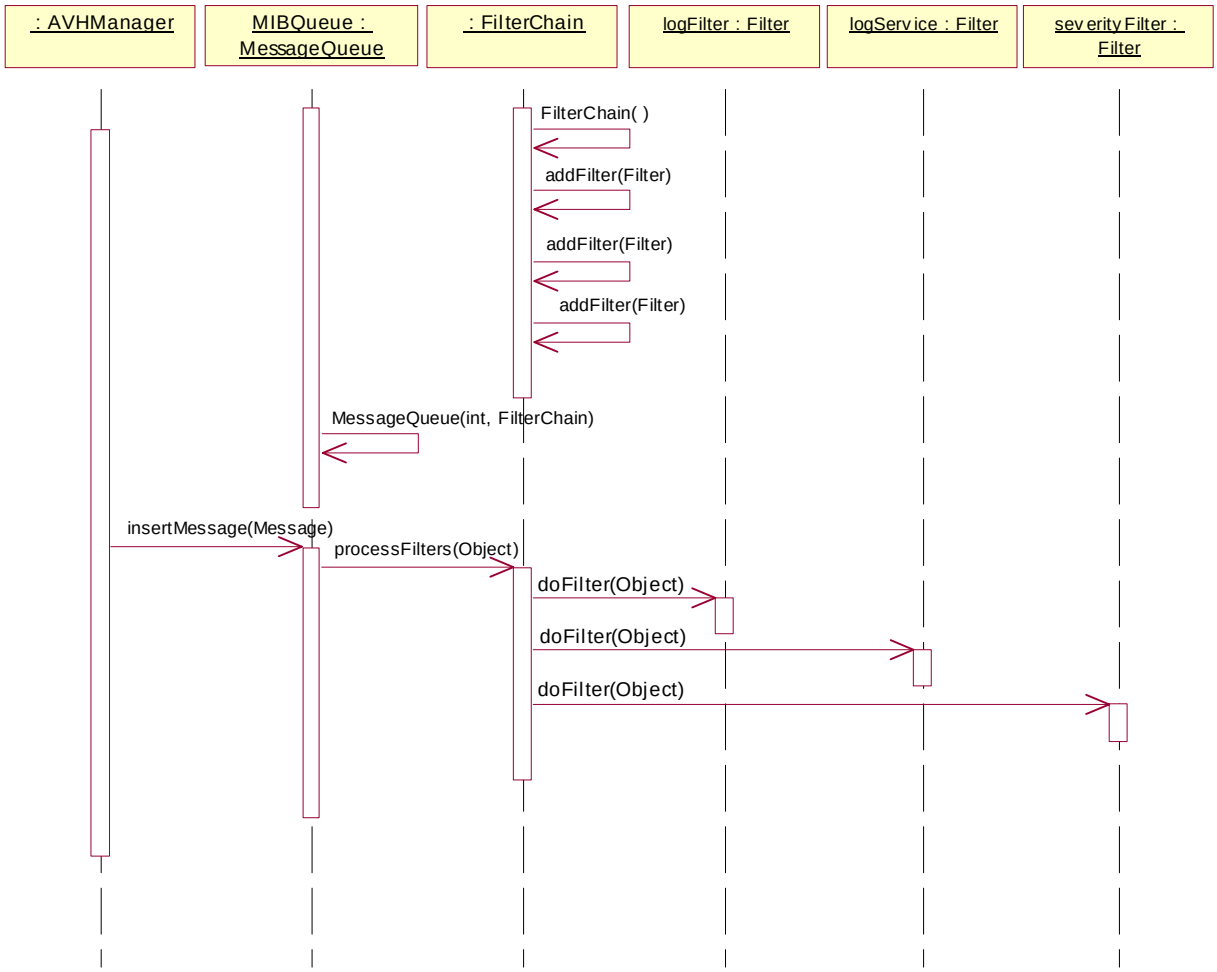


Figura 6.11 Modelo Dinámico de la Solución de Diseño a los Conceptos "*Filtro de Procesamiento de Bitácora*", "*Servicio Analizador de Severidad de Alarma*" y "*Servicio de Bitácora*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. Se inicia el mecanismo de filtrado de mensajes definiendo los filtros que deben aplicarse. Esto se realiza en el momento en que se configura la aplicación, utilizando el método *addFilter(…)* de *FilterChain*. En este caso se definieron dos filtros.
2. Cuando existe un cambio de estado en un Recurso, la instancia de *MIBManager* genera un *ReportMessage* y llama al método *insertMessage(…)* de *mibQueue*.
3. *mibQueue* a su vez manda llamar el método *processFilters(…)* de *FilterChain* para que le sean aplicados los filtros al mensaje.

4. Se aplican los filtros a través de la llamada *doFilter(…)* de las instancias de *Filter* definidas para la aplicación de discriminantes en el mensaje. En este caso se aplican los filtros *logFilter*, *logService* y *severityFilter*.

6.7 Diseño del Componente "AVH"

El diseño de la solución para el componente AVH debe cumplir con los siguientes objetivos:

1. **Monitoreo Constante a los Recursos**. Debe existir el monitoreo constante en los Recursos, lo cual permita vigilar el cambio de estado en los mismos y se refleje en los Objetos Administrados, de tal forma que si en algún momento el cambio de estado repercute en la funcionalidad del Elemento de Red, esto sea reportado al Administrador.
2. **Obtención del Estado de los Recursos**. La obtención del estado en los Recursos sirve para el monitoreo de los mismos. Esto se debe efectuar independientemente de los Recursos con los que cuente el Elemento de Red.

Una vez determinados los objetivos específicos que el framework deberá cumplir con el diseño del componente "*AVH*", se presenta a continuación para cada concepto asociado con el componente, su responsabilidad, los puntos comunes y de variabilidad determinados en el AVCD y la solución a éstos.

6.7.1 Administrador Virtual de Hardware, Buscador de Recursos, Lista de Mapeo de Recursos, Recurso, Reporte de Cambio de Estado

El diseño de la solución de los conceptos *Administrador Virtual de Hardware, Buscador de Recursos, Lista de Mapeo de Recursos, Recurso, Reporte de Cambio de Estado* se presenta de manera agrupada debido a que éstos representan la funcionalidad conjunta de la Administración Virtual de Hardware, por lo que es mejor presentar el diseño de éstos conceptos de manera conjunta a presentarlos de manera individual.

6.7.1.1 Responsabilidad

- El *Administrador Virtual de Hardware* se encarga de administrar y monitorear los Recursos de un elemento de Red, además genera los reportes de cambio de estado pertinentes, los cuales deben ser tratados posteriormente por la *MIB*.
- El *Buscador de Recursos* permite obtener dado un identificador, el *Recurso* solicitado.
- La *Lista de Mapeo de Recursos* tiene como responsabilidad el relacionar a los Recursos con los Objetos Administrados.
- El *Recurso* tiene la lógica necesaria para obtener el estado de un recurso de hardware.
- El *Reporte de Cambio de Estado* encapsula el evento de cambio de estado de un *Recurso* de hardware.

6.7.1.2 Puntos Comunes

Los puntos comunes del concepto *Administrador Virtual de Hardware* son:

- Servicio común en los elementos de Red, el cual monitorea los recursos de hardware del elemento. El *Administrador Virtual de Hardware* es independiente del modelo de información utilizado.

Los puntos comunes del concepto *Buscador de Recursos* son:

- Servicio común de búsqueda de Recursos ya sea por su identificador o por el identificador del Objeto Administrado.

Los puntos comunes del concepto *Lista de Mapeo de Recursos* son:

- Relaciona a los Recursos de Hardware con los Objetos Administrados.

Los puntos comunes del concepto *Recurso* son:

- Elemento que contiene información y la lógica necesaria para obtener el estado de un Recursos de hardware.

Los puntos comunes del concepto *Reporte de Cambio de Estado* son:

- Contiene información común que reporta el cambio de estado en un Recurso.

6.7.1.3 Puntos de Variabilidad

Los puntos de variabilidad del concepto *Administrador Virtual de Hardware* son:

- Tipo de monitoreo aplicado a los Recursos.
- Diversidad de Recursos a monitorear.

Los puntos de variabilidad del concepto *Buscador de Recursos* son:

- Diversidad de Identificadores para la búsqueda de Recursos.
- Información utilizada para la búsqueda de Recursos.

La *Lista de Mapeo de Recursos* no tiene ningún punto de variabilidad.

Los puntos de variabilidad del concepto *Recurso* son:

- Diversidad de representaciones de un Recurso de Elemento de Red, además de las formas en que se puede obtener el estado de cada uno de ellos.

Los puntos de variabilidad del concepto *Reporte de Cambio de Estado* son:

- Diversidad de información y formatos de reportes.

6.7.1.4 Solución

El Administrador Virtual de Hardware debe encargarse del monitoreo de los Recursos de una forma constante cada cierto intervalo de tiempo, lo cual es un comportamiento *común* en esta entidad de software, no obstante la *variabilidad* se encuentra en la forma en que este elemento puede aplicar el monitoreo, ya que puede ser de distintas formas, por lo cual el monitoreo debe ser abstracto en el framework y concretizado al desarrollar la aplicación del Agente.

Debido a que el Administrador Virtual de Hardware debe mantener referencia a los Recursos, éste debe tener un contenedor, el cual permita iterar los Recursos para efectos de monitoreo.

La Lista de Mapeo de los Recursos se implementa a través del Buscador de Recursos, ya que es éste el que permite obtener un Recurso dado los identificadores del Objeto Administrado.

Los Recursos deben mantener abstracta la forma en que se obtiene su estado, encapsulando las funciones de bajo nivel para este efecto. Los Recursos también deben contar con el identificador del Objeto Administrado con el que están relacionados, esto con el fin de facilitar las búsquedas entre estos elementos y permitir la implementación del mecanismo de mapeo entre Recursos y Objetos Administrados.

Por último los Reportes de Cambio de Estado deben heredar las propiedades del Mensaje, para que en caso de que éste llegue al Agente, le facilite identificar el servicio que atiende a los Reportes de Cambio de Estado.

6.7.1.5 Modelo Estático

El modelo estático de los conceptos *Administrador Virtual de Hardware, Buscador de Recursos, Lista de Mapeo de Recursos, Recurso, Reporte de Cambio de Estado* consiste en la aplicación de los patrones de diseño "*Access Proxy*" [Gamma, 1995], *Observer* [Gamma, 1995], *Iterator* [Gamma, 1995], *Strategy* [Gamma, 1995] y *Adapter* [Gamma, 1995] los cuales se reflejan en diagrama de clases de la Figura 6.12.

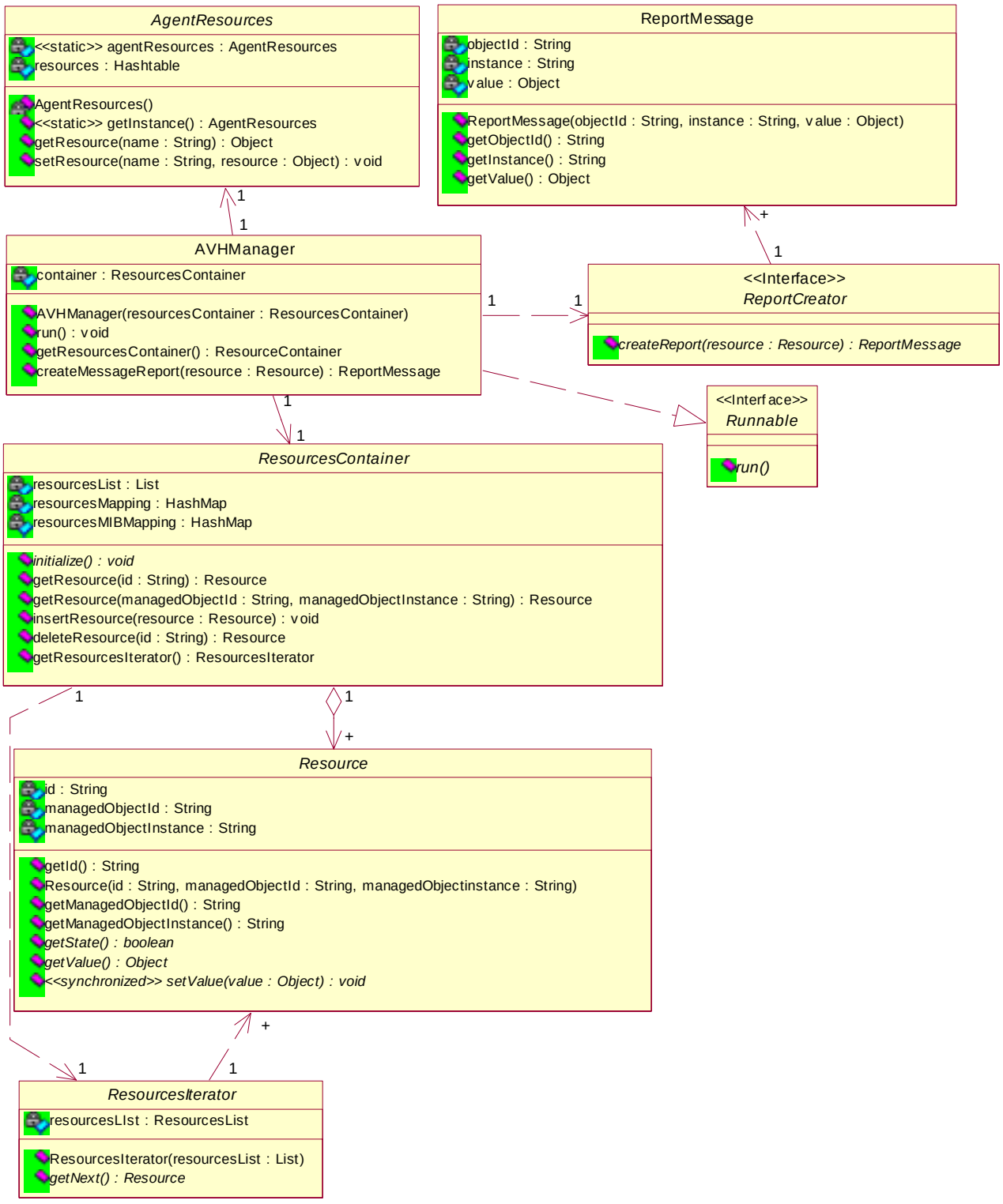


Figura 6.12 Modelo Estático de la Solución de Diseño a los Conceptos "*Administrador Virtual de Hardware*", "*Buscador de Recursos*", "*Lista de Mapeo de Recursos*", "*Recurso*", "*Reporte de Cambio de Estado*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Access Proxy***. Este patrón de diseño es implementado a través de los métodos *getResource(…), insertResource(…)* y *deleteResource(…),* ya que permiten o niegan el acceso a los métodos del contenedor *ResourceContainer*, y es solo a través de éstos métodos que se puede acceder a la funcionalidad del contenedor de Objetos Administrados.

- ***Observer***. Este patrón de diseño es implementado a través del método principal del *AVHManager* (*run(…)*)ya que se encarga de observar constantemente el estado de los Recursos y en caso de que el estado no sea favorable, entonces se encarga de reportarle al *MIBManager* de este suceso a través del envío del reporte a la *MessageQueue* correspondiente.

- ***Iterator***. Este patrón de diseño es reflejado en el la clase *ResourcesIterator*, ya que le permite al *AVHManager* obtener el siguiente elemento de la lista de *Resource*. Cabe destacar que el método *getNext()* de *ResoucesIterator* es abstracto debido a que no se sabe el orden de monitoreo que se debe implementar, ya que puede ser aleatorio, secuencial, por prioridades, etc.

- ***Strategy***. Este patrón de diseño es reflejado en el método *createReport(…),* ya que el *AVHManager* delega a la instancia de *ReportCreator* la responsabilidad de crear el reporte de cambio de estado de un Recurso. El método *createReport(…)* es abstracto, por lo cual el algoritmo para crear el reporte de cambio de estado es responsabilidad del desarrollador del Agente.

- ***Adapter***. Este patrón de diseño es implementado en el método *setValue(…), getValue(…)* y *getState(…)* de la clase *Resource*, ya que de manera abstracta estos métodos deberá ser implementado por subclases de Recursos, los cuales tengan la lógica necesaria para adaptar estos métodos de acuerdo a sus necesidades y formatos de información que puedan recibir para modificar el estado del Objeto Administrado, obtener su valor, y obtener el estado actual del mismo, el cual permita verificar si es necesario reportar un cambio de estado crítico.

6.7.1.6 Descripción de Clases

La descripción de las clases *AVHManager, ResourcesContainer, ResourcesIterator, Resource, ReportCreator* y *ReportMessage* se encuentra definida en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.4 Descripción de Clases del Componente AVH*".

6.7.1.7 Lógica de Control del Framework

La lógica de control de la capa AVH se encuentra en el método *run(…)* se encuentra reflejada en el método *insertMessage(…)* de la clase *AVHManager* ya contiene la lógica necesaria para efectuar el monitoreo sobre los Recursos del Elemento de Red. Por lo tanto este método no deben ser redefinido a menos que se requiera modificar la lógica de monitoreo de los Recursos del elemento de Red.

6.7.1.8 Extensibilidad

A continuación se presentan una guía para la extensibilidad del framework en las clases *ResourcesContainer*, *Resource*, *ResourcesIterator* y *ReportCreator*.

ResourcesContainer. Clase encargada de contener los Recursos pertenecientes a un Elemento de Red. Los métodos que deben ser concretizados son los siguientes:

- ***initialize(…)***. Inicializa el contenedor de los Recursos del Elemento de Red. Este método es abstracto debido a que no se sabe con exactitud que actividades se deben realizar en el momento de poner disponible al contenedor de Recursos. El comportamiento esperado de este método es que realice las actividades previas a la disponibilidad del contenedor, generando las instancias de Recursos y contenerlos en la estructura de datos definida.

Resource. Clase que encapsula la información referente a un Recurso de Elemento de Red. Los métodos que deben ser concretizados son:

- ***setValue(…)***. Cambia el valor o estado del Recurso. Este método es abstracto debido a que no se sabe con exactitud las actividades que se deben realizar al momento de cambiar el estado del Recurso, ya que los Recursos pueden ser de Hardware o software y la forma en que se modifica el estado puede ser muy variable. El comportamiento esperado es que se actualice el estado del Recurso y en caso de que no sea adecuado el cambio de estado el método debe retornar *false*, en caso contrario se retorna *true*.
- ***getValue(…)***. Obtiene el valor actual del Recurso. Este método es abstracto debido a que no se sabe con exactitud el tipo de valor que pueda obtenerse a parte de que el mecanismo de obtención del valor puede variar debido a que existen Recursos de hardware o de softsoware. El comportamiento esperado es que se obtenga el valor del Recurso utilizando el mecanismo que sea requerido para realizarlo.
- ***getState(…)***. Verifica el estado actual del Recurso. Este método es abstracto debido a que no se sabe con exactitud las actividades que se deben realizar al momento de obtener el estado actual del Recurso, ya que los Recursos pueden ser de Hardware o software y la forma en que se obtiene el estado puede ser muy variable. El comportamiento esperado es que se obtenga el estado del Recurso y en caso de que éste haya cambiado respecto de su estado anterior, se debe retornar *false*, en caso contrario se retorna *true*.

ResourcesIterator. Clase que contiene la lógica de iteración en el contenedor de Recursos para que el Administrador Virtual de Hardware pueda monitorearlos Los métodos que deben ser concretizados son:

- ***getNext(…)***. Obtiene el siguiente Recurso del contenedor *AVHContainer* para obtener su estado, y en su caso generar un reporte de cambio de estado. Cambia el valor o estado del Recurso. Este método es abstracto debido a que no se sabe con exactitud que mecanismo se utilice para obtener el siguiente Recurso a monitorear, ya que se puede establecer por prioridad, aleatoriamente, secuencial, etc. El comportamiento esperado es que se obtenga el siguiente Recurso a ser monitoreado, dado el mecanismo de secuencia establecido.

ReportCreator. Clase que permite crear un reporte de cambio de estado en un Recurso. Los métodos que deben ser concretizados son:

- ***createReport(…)***. Crea un reporte de cambio de estado en un Recurso. Este método es abstracto debido a la diversidad de Recursos que puede existir, pudiendo utilizar para cada uno de ellos diferentes formatos de reporte. El comportamiento esperado es que dado un Recurso, este método genere un *ReportMessage* adecuado para el Recurso.

6.7.1.9 Modelo Dinámico

La Figura 6.13(a) y Figura 6.13(b) presentan el modelo dinámico entre instancias de las *AVHManager*, *ResourcesContainer*, *ResourcesIterator*, *Resource*, *ReportCreator* y *ReportMessage*.

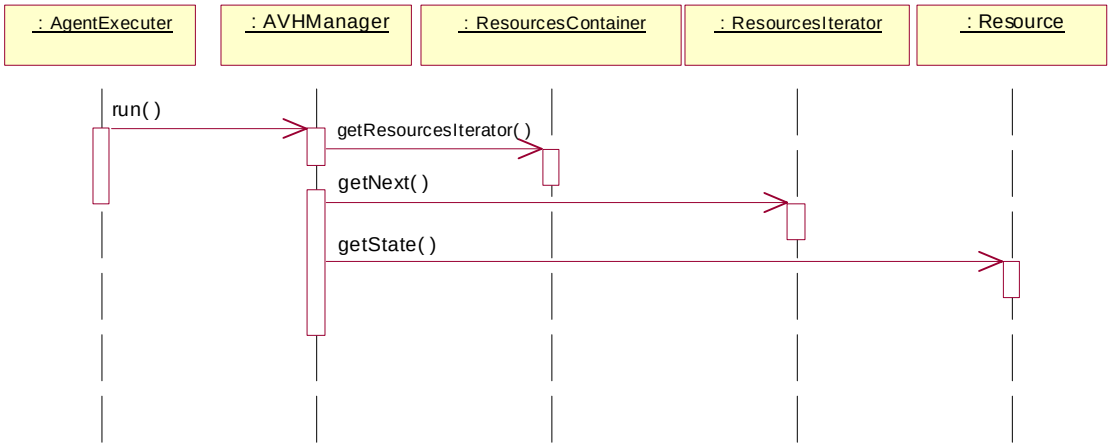


Figura 6.13(a) Modelo Dinámico de la Solución de Diseño a los Conceptos "*Administrador Virtual de Hardware*", "*Buscador de Recursos*", "*Lista de Mapeo de Recursos*", "*Recurso*", "*Reporte de Cambio de Estado*".

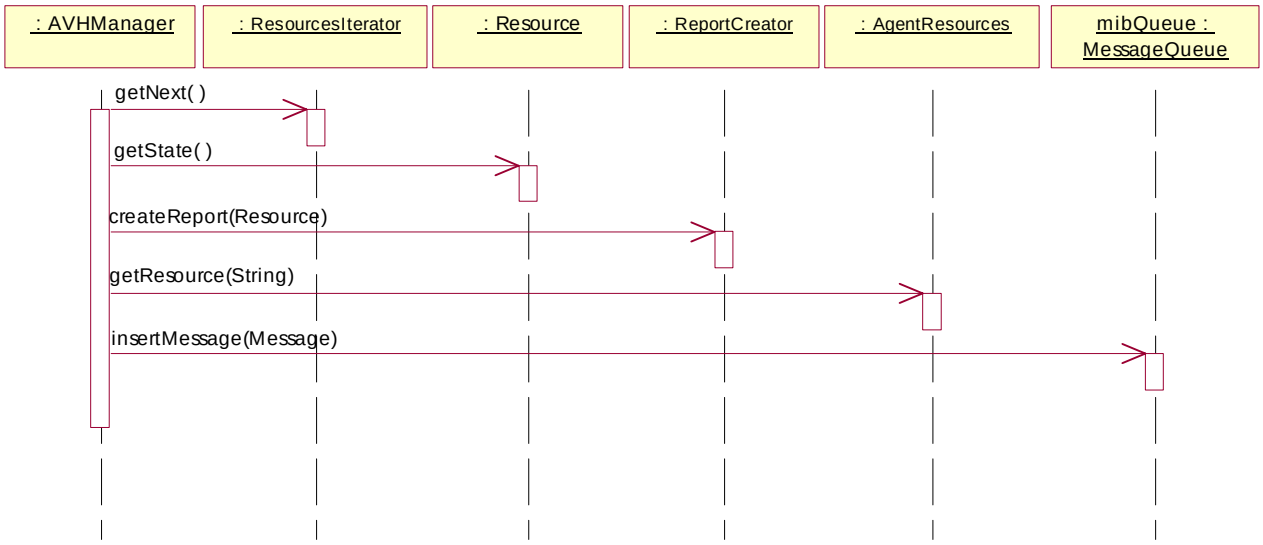


Figura 6.13(b) Modelo Dinámico de la Solución de Diseño a los Conceptos "*Administrador Virtual de Hardware*", "*Buscador de Recursos*", "*Lista de Mapeo de Recursos*", "*Recurso*", "*Reporte de Cambio de Estado*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. Se inicia el mecanismo de monitoreo de los Recursos cuando la instancia de la clase *AVHManager* obtiene un objeto de la clase *ResourceIterator*, el cual es proporcionado al llamar a la función *getResourceIterator(…)* de la clase *MIBContainer*.
2. Se obtiene el siguiente Recurso del cual se debe vigilar su estado, esto se realiza al llamar a la función *getNext(…)* de *ResourceIterator*.
3. Al obtener el Recurso se verifica que su estado no haya cambiado, realizando una llamada a la función *getState(…)* de *Resource*.
4. En caso de que el estado del Recurso haya cambiado, entonces se crea un reporte de cambio de estado llamando a la función *createReport(…)* de *ReportCreator*.
5. La instancia de *AVHManager* obtiene la referencia a la cola de mensajes de la MIB, realizando una llamada a la función *getResource(…)* de la clase *AgentResources*.
6. Por último se inserta el reporte en la cola de mensajes de la MIB al llamar a la función *insertMessage(…)* de *mibQueue*. El reporte entonces es tratado por el Administrador de la MIB y en su caso atendido por el Agente para que sea enviado al Administrador.

6.8 Configuración del Framework

La configuración del framework debe cumplir con los siguientes objetivos:

1. **Configuración de Servicios del Agente**. La configuración del framework debe determinar los servicios y/o comandos que el Agente soportará, relacionándolos con los mensajes enviados por el Administrador.
2. **Determinación de Recursos disponibles a los Servicios del Agente**. El configurador del framework determinará los recursos que están disponibles para la ejecución de los servicios del Agente, como las colas de espera de mensajes del Agente, el tipo de contenedor utilizado en la MIB, etc.
3. **Configuración a través de XML**. EL uso de XML para la configuración de servicios y recursos permite modificar de manera fácil las variabilidades encontradas en la funcionalidad del Agente, las cuales dependen del Elemento de Red, sus capacidades de hardware y software, los Recursos, etc.

Una vez determinados los objetivos específicos del C*onfigurador del Framework*, se presenta a continuación la solución utilizada para proporcionar la funcionalidad del mismo, el modelo estático, los patrones de diseño utilizados y el modelo dinámico.

6.8.1 Configurador del Framework

6.8.1.1 Solución

El configurador del framework debe utilizar un esquema que permita ser rígido al definir las los elementos que deben estar presentes en cualesquier aplicación y a la vez flexible al permitir definir diversos servicios, recursos y clases que puedan ser utilizados en las aplicaciones que extiendan al framework.

Es por ello que se opta por utilizar un esquema XML, el cual tiene secciones en donde se deben definir los elementos siempre presentes en la capa de Agente, MIB y AVH, y secciones en donde se ponen disponibles recursos y clases que pueden ser utilizados en la funcionalidad integral de la aplicación Agente.

Al iniciar la configuración y comenzar la ejecución del Agente, el configurador de la aplicación busca el archivo XML y lo analiza, obteniendo de esta forma las clases concretas que extienden de clases del framework, lo cual permite definir las características variables del framework y a su vez a la aplicación en sí. Estas clases y recursos se ponen disponibles a través de contenedores, los cuales pueden ser accedidos fácilmente por cualquier elemento del framework.

Una vez que se obtienen las clases que extienden del framework, se inicia la etapa de ejecución de la aplicación, creando cuatro hilos de control, el receptor de mensajes, el Agente, el Administrador de la MIB y el Administrador de la AVH. Una vez iniciados estos cuatro elementos, la operación normal del Agente está en funciones, listo para recibir mensajes, analizarlos y ejecutarlos en su caso, y para atender los cambios de estado en Recursos y Objetos Administrados.

El DTD y un ejemplo de configuración XML del framework se presentan a continuación.

6.8.1.2 DTD de Configuración

```
<?xml version="1.0"?>
<!DOCTYPE network-element [                  <!-- network-element (Raiz del configurador).    -->
<!ELEMENT network-element (agent, mib, avh)> <!-- agent, mib y avh componen a network-agent.  -->
<!ATTLIST network-element                    <!-- Lista de atributos de network-element:      -->
          id          CDATA #REQUIRED        <!-- *  Identificador del Elemento de Red.       -->
          ip          CDATA #REQUIRED        <!-- *  IP del Elemento de Red.                  -->
          name        CDATA #REQUIRED        <!-- *  Nombre del Elemento de Red.              -->
          description CDATA #REQUIRED>       <!-- *  Descripción del Elemento de Red.         -->
                                             <!-- agent (Capa de Agente).                     -->
                                             <!-- agent se compone de los elementos:          -->
<!ELEMENT agent (agent-mandatory-resources,  <!-- *  agent-mandatory-resources                -->
                 agent-additional-resources, <!-- *  agent-additional-resources               -->
                 agent-additional-classes)>  <!-- *  agent-additional-classes                 -->

                                             <!-- En el elemento agent-mandatory-resources se -->
                                             <!-- definen Recursos Obligatorios del Agente,   -->
<!ELEMENT agent-mandatory-resources          <!-- los cuales son los siguientes:              -->
        (message-decoder, message-receptor,  <!-- *  message-decoder, message-receptor,       -->
         agent-queue, agent-filter-chain,    <!-- *  agent-queue, agent-filter-chain          -->
         process-manager, services,          <!-- *  process-manager, services                -->
         message-transmisor,                 <!-- *  message-transmisor                       -->
         message-coder, administrators)>     <!-- *  message-coder, administrators            -->
```

```
                                    <!-- En el elemento agent-additional-resources   -->
                                    <!-- se definen Recursos Adicionales del Agente. -->
<!ELEMENT agent-additional-resources <!-- agent-additional-resources se compone de:   -->
          (resource*)>              <!-- *  resource  (0 o más recursos)             -->

                                    <!-- En el elemento agent-additional-classes     -->
                                    <!-- se definen Clases Adicionales del Agente.   -->
<!ELEMENT agent-additional-classes  <!-- agent-additional-clasess se compone de:     -->
          (add-class*)>             <!-- *  add-class (0 o más clases)               -->

                                    <!-- mib (Capa Management Information Base).     -->
<!ELEMENT mib                       <!-- mib se compone de los elementos:            -->
          (mib-mandatory-resources, <!-- *  mib-mandatory-resources                 -->
           mib-additional-resources,<!-- *  mib-additional-resources                -->
           mib-additional-classes)> <!-- *  mib-additional-classes                  -->

                                    <!-- En el elemento mib-mandatory-resources se   -->
                                    <!-- definen Recursos Obligatorios de la MIB,    -->
<!ELEMENT mib-mandatory-resources   <!-- los cuales son los siguientes:              -->
          (mib-container,           <!-- *  mib-container                           -->
           mib-queue,               <!-- *  mib-queue                               -->
           filter-chain)>           <!-- *  filter-chain                            -->

                                    <!-- En el elemento mib-additional-resources     -->
                                    <!-- se definen Recursos Adicionales de la MIB.  -->
<!ELEMENT mib-additional-resources  <!-- mib-additional-resources se compone de:     -->
          (resource*)>              <!-- *  resource (0 o más recursos)              -->

                                    <!-- En el elemento mib-additional-classes       -->
                                    <!-- se definen Clases Adicionales de la MIB.    -->
<!ELEMENT mib-additional-classes    <!-- mib-additional-classes se compone de:       -->
          (add-class*)>             <!-- *  add-class (0 o más clases)               -->

                                    <!-- avh (Capa de Admón. Virtual de Hardware).   -->
<!ELEMENT avh                       <!-- avh se compone de los elementos:            -->
          (avh-mandatory-resources, <!-- *  avh-mandatory-resources                 -->
           avh-additional-resources,<!-- *  avh-additional-resources                -->
           avh-additional-classes)> <!-- *  avh-additional-clases                   -->

                                    <!-- En el elemento avh-mandatory-resources se   -->
                                    <!-- definen Recursos Obligatorios de la AVH,    -->
<!ELEMENT avh-mandatory-resources   <!-- los cuales son los siguientes:              -->
          (avh-manager,             <!-- *  avh-manager                             -->
           resources-container,     <!-- *  resources-container                     -->
           resources-iterator,      <!-- *  resources-iterator                      -->
           report-creator)>         <!-- *  report-creator                          -->

                                    <!-- En el elemento avh-additional-resources     -->
                                    <!-- se definen Recursos Adicionales de la AVH.  -->
<!ELEMENT avh-additional-resources  <!-- avh-additional-resources se compone de:     -->
          (resource*)>              <!-- *  resource (0 a más recursos)              -->

                                    <!-- En el elemento avh-additional-classes       -->
                                    <!-- se definen Clases Adicionales de la AVH.    -->
<!ELEMENT avh-additional-classes    <!-- avh-additional-classes se compone de:       -->
          (add-class*)>             <!-- *  add-class (0 o más clases)               -->

                                    <!-- message-receptor es el receptor de mensajes -->
<!ELEMENT message-receptor EMPTY>   <!-- del Agente.                                 -->
<!ATTLIST message-receptor          <!-- Lista de atributos de message-receptor:     -->
          class CDATA #REQUIRED     <!-- *  Clase que representa a message-receptor, -->
                                    <!--    debe extender de MessageReceptor.       -->
          port  CDATA #REQUIRED>    <!-- *  Puerto para la recepción de mensajes.    -->

                                    <!-- message-decoder decodifica un flujo de inf. -->
<!ELEMENT message-decoder EMPTY>    <!-- y lo transforma en instancias de Message.   -->
<!ATTLIST message-decoder           <!-- Lista de atributos de message-decoder:      -->
          class CDATA #REQUIRED>    <!-- *  Clase que representa a message-decoder,  -->
                                    <!--    debe extender de MessageDecoder.        -->
```

```
                                        <!-- agent-queue contiene los mensajes enviados  -->
<!ELEMENT agent-queue EMPTY>            <!-- por el Administrador.                       -->
<!ATTLIST agent-queue                   <!-- Lista de atributos de agent-queue:          -->
          class    CDATA #REQUIRED      <!-- *  Clase que representa a agent-queue,      -->
                                        <!--    debe extender de MessageQueue.           -->
          capacity CDATA #REQUIRED>     <!-- *  Capacidad máxima en elementos de la cola -->
                                        <!--    de mensajes.                             -->

                                        <!-- agent-filter-chain contiene los filtros que -->
                                        <!-- se deben aplicar al intentar insertar un    -->
<!ELEMENT agent-filter-chain            <!-- mensaje en agent-queue. Se compone de:      -->
          (filter*)>                    <!-- *  filter (0 o más filtros.)                -->

                                        <!-- process-manager contiene un pool de process -->
<!ELEMENT process-manager EMPTY>        <!-- quienes se utilizan al ejecutar un service. -->
<!ATTLIST process-manager               <!-- Lista de atributos de process-manager:      -->
          class    CDATA #REQUIRED      <!-- *  Clase que representa al Administrador de -->
                                        <!--    procesos, debe extender de ProcessManager-->
          capacity CDATA #REQUIRED>     <!-- *  Capacidad máxima del pool de Procesos    -->

                                        <!-- services define los servicios que el Agente -->
                                        <!-- soportará para la ejecución de comandos     -->
                                        <!-- solicitados por el Administrador.           -->
<!ELEMENT services                      <!-- services se compone de:                     -->
          (service+)>                   <!-- * service (1 o más servicios)               -->

                                        <!-- service define un servicio que se relaciona -->
                                        <!-- con un comando del protocolo de admón. de   -->
<!ELEMENT service EMPTY>                <!-- red utilizado.                              -->
<!ATTLIST service                       <!-- Lista de atributos de service:              -->
          command-id   CDATA #REQUIRED      <!-- Identificador binario del comando          -->
          command-text CDATA #REQUIRED  <!-- Identificador textual del comando del       -->
                                        <!-- protocolo de Administración de Red          -->
                                        <!-- representado por el servicio.               -->
          class      CDATA #REQUIRED>   <!-- *  Clase que representa al Servicio         -->
                                        <!--    identificado por command-id, debe        -->
                                        <!--    extender de Service                      -->

                                        <!-- message-transmisor es el transmisor de      -->
<!ELEMENT message-transmisor EMPTY>     <!-- mensajes del Agente.                        -->
<!ATTLIST message-transmisor            <!-- Lista de atributos de message-transmisor:   -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa a message-transmisor-->
                                        <!--    debe extender de MessageTransmisor.      -->

                                        <!-- message-coder codifica una instancia de     -->
                                        <!-- Message y lo transforma en un flujo de      -->
<!ELEMENT message-coder EMPTY>          <!-- datos.                                      -->
<!ATTLIST message-coder                 <!-- Lista de atributos de message-coder:        -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa al message-coder    -->
                                        <!--    debe implementar a MessageCoder.         -->

                                        <!-- administrators define un conjunto de       -->
                                        <!-- Administradores a los cuales se les debe    -->
                                        <!-- enviar mensajes de retorno.                 -->
<!ELEMENT administrators                <!-- administrators se compone de:               -->
          (administrator+)>             <!-- *  administrator                            -->

                                        <!-- administrator define un Administrador al que-->
<!ELEMENT administrator EMPTY>          <!-- se deben enviar mensajes de retorno         -->
<!ATTLIST administrator                 <!-- Lista de atributos de administrator:        -->
          ip    CDATA #REQUIRED         <!-- IP del Administrador                        -->
          port  CDATA #REQUIRED>        <!-- Puerto de entrada del Administrador         -->

                                        <!-- mib-container representa el contenedor de   -->
<!ELEMENT mib-container EMPTY>          <!-- Objetos Administrados de la MIB.            -->
<!ATTLIST mib-container                 <!-- Lista de atributos de mib-container:        -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa al mib-container,   -->
                                        <!--    debe implementar a MIBContainer.         -->
```

```
                                        <!-- mib-filter-chain contiene los filtros que   -->
                                        <!-- se deben aplicar al intentar insertar un    -->
<!ELEMENT mib-filter-chain              <!-- mensaje en mib-queue. Se compone de:        -->
          (filter*)>                    <!-- *  filter (0 o más filtros.)                -->

                                        <!-- filter define un filtro que se debe aplicar -->
<!ELEMENT filter EMPTY>                 <!-- a un elemento, o en este caso a un Message. -->
<!ATTLIST filter                        <!-- Lista de atributos de filter:               -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa al filtro,          -->
                                        <!--    debe implementar a la interfaz Filter.   -->

                                        <!-- mib-queue contiene los mensajes de reporte  -->
<!ELEMENT mib-queue EMPTY>              <!-- de cambio de estado enviados por el AVH     -->
<!ATTLIST mib-queue                     <!-- Lista de atributos de mib-queue:            -->
          class    CDATA #REQUIRED      <!-- *  Clase que representa a mib-queue,        -->
                                        <!--    debe extender a MessageQueue.            -->
          capacity CDATA #REQUIRED>     <!-- *  Capacidad máxima en elementos de la cola -->
                                        <!--    de mensajes.                             -->

                                        <!-- avh-manager representa el Administrador     -->
<!ELEMENT avh-manager EMPTY>            <!-- Virtual de Hardware.                        -->
<!ATTLIST avh-manager                   <!-- Lista de atributos de avh-manager:          -->
          sleep-time CDATA #REQUIRED>   <!-- *  Tiempo en milisegundos que debe esperar  -->
                                        <!--    al AVHManager para obtener el siguiente  -->
                                        <!--  Recurso del cual se va a analizar su estado-->

                                        <!-- resources-container representa el contenedor-->
<!ELEMENT resources-container EMPTY>    <!-- de Recursos de la AVH.                      -->
<!ATTLIST resources-container           <!-- Lista de atributos de resources-container:  -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa a resource-container-->
                                        <!--    debe extender a ResourcesContainer.      -->

                                        <!-- resources-iterator representa al iterador   -->
                                        <!-- de recursos utilizado por el AVHManager para-->
<!ELEMENT resources-iterator EMPTY>     <!-- el monitoreo de los mismos.                 -->
<!ATTLIST resources-iterator            <!-- Lista de atributos de resources-iterator:   -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa a resource-iterator -->
                                        <!--    debe extender a ResourcesIterator.       -->

                                        <!-- report-creator representa al creador de     -->
<!ELEMENT report-creator EMPTY>         <!-- mensajes de cambio de estado en los Recursos-->
<!ATTLIST report-creator                <!-- Lista de atributos de report-creator:       -->
          class CDATA #REQUIRED>        <!-- *  Clase que representa a report-creator    -->
                                        <!--    debe implementar a ReportCreator.        -->

                                        <!-- resource representa un recurso adicional que-->
<!ELEMENT resource EMPTY>               <!-- se desea dejar disponible para su uso en    -->
                                        <!-- el contenedor AgentResources                -->
<!ATTLIST resource                      <!-- Lista de atributos de resource:             -->
          key   CDATA #REQUIRED         <!-- Clave de Recurso que permite identificarlo. -->
          class CDATA #REQUIRED>        <!-- Clase que representa al recurso             -->

                                        <!-- add-class representa una clase adicional que-->
<!ELEMENT add-class EMPTY>              <!-- se desea dejar disponible para su uso en    -->
                                        <!-- el contenedor ClassesLoader                 -->
<!ATTLIST add-class                     <!-- Lista de atributos de add-class:            -->
          key   CDATA #REQUIRED         <!-- Clave de la Clase que permite identificarla.-->
          class CDATA #REQUIRED>        <!-- Clase.                                      -->

]>                                      <!-- Fin del DTD                                 -->
```

6.8.1.3 Ejemplo de Archivo XML de Configuración

```
<network-element id="NET09AK783909900" ip="200.34.38.191" name="Ruteador CISCO A9000"
                 description="Ruteador CISCO A9000">
   <agent>
      <agent-mandatory-resources>
         <message-decoder       class="com.networkagent.networkelement.agent.UDPDecoder"/>
         <message-receptor      class="com.networkagent.networkelement.agent.UDPReceptor"
                                port="161"/>
         <agent-filter-chain>
            <filter             class="com.networkagent.networkelement.agent.DecripterFilter"/>
            <filter             class="com.networkagent.networkelement.agent.UnzipperFilter"/>
         </agent-filter-chain>
         <agent-queue           class="com.networkagent.networkelement.agent.Queue"
                                capacity="100"/>
         <process-manager       class="com.networkagent.networkelement.agent.AgentPM"
                                capacity="50"/>
         <services>
            <service            command-id="a0" command-text="GET-REQUEST"
                                class="com.networkagent.networkelement.agent.GetRequestService"/>
            <service            command-id="a4" command-id="TRAP"
                                class="com.networkagent.networkelement.agent.TrapService"/>
         </services>
         <message-transmisor    class="com.networkagent.networkelement.agent.UDPTransmisor"/>
         <message-coder         class="com.networkagent.networkelement.agent.UDPCoder"/>
         <administrators>
            <administrator      ip="200.34.38.190"  port="164"/>
         </administrators>
      </agent-mandatory-resources>
      <agent-additional-resources>
         <resource              key="agent-logger"
                                class="com.networkagent.networkelement.agent.AgentLogger"/>
      </agent-additional-resources>
      <agent-additional-classes/>
   </agent>
   <mib>
      <mib-mandatory-resources>
         <mib-container         class="com.networkagent.networkelement.mib.DBContainer"/>
         <mib-filter-chain>
            <filter             class="com.networkagent.networkelement.mib.LoggerFilter"/>
         </mib-filter-chain>
         <mib-queue             class="com.networkagent.networkelement.mib.Queue"
                                capacity="100"/>
      </mib-mandatory-resources>
      <mib-additional-resources/>
      <mib-additional-classes/>
   </mib>
   <avh>
      <avh-mandatory-resources>
         <avh-manager           sleep-time="2000"/>
         <resources-container   class="com.networkagent.networkelement.avh.ListContainer"/>
         <resources-iterator    class="com.networkagent.networkelement.avh.RandomIterator"/>
         <report-creator        class="com.networkagent.networkelement.avh.TrapReportCreator"/>
      </avh-mandatory-resources><avh-additional-resources/><avh-additional-classes/>
   </avh>
</network-element>
```

En el XML de ejemplo se puede observar como se van configurando las clases necesarias para que el Agente cumpla con su función. Los recursos obligatorios en cada una de las capas deben heredar de las clases correspondientes definidas en el framework, por ejemplo se puede observar que se definen dos servicios que el Agente puede atender, "*GET-REQUEST*" y "*TRAP*", cuyas respectivas clases son *com.networkagent.networkelement.agent.GetRequest* y *com.networkagent.networkelement .agent.Trap*, las cuales deben heredar de la clase *Service*, definida en el framework.

6.8.1.4 Modelo Estático

El modelo estático del *Configurador del Framework* consiste en la aplicación de los patrones de diseño "*Factory Method*" [Gamma, 1995] y *Dynamic Linkage* [Grand, 1998], los cuales se reflejan en diagrama de clases de la Figura 6.14.

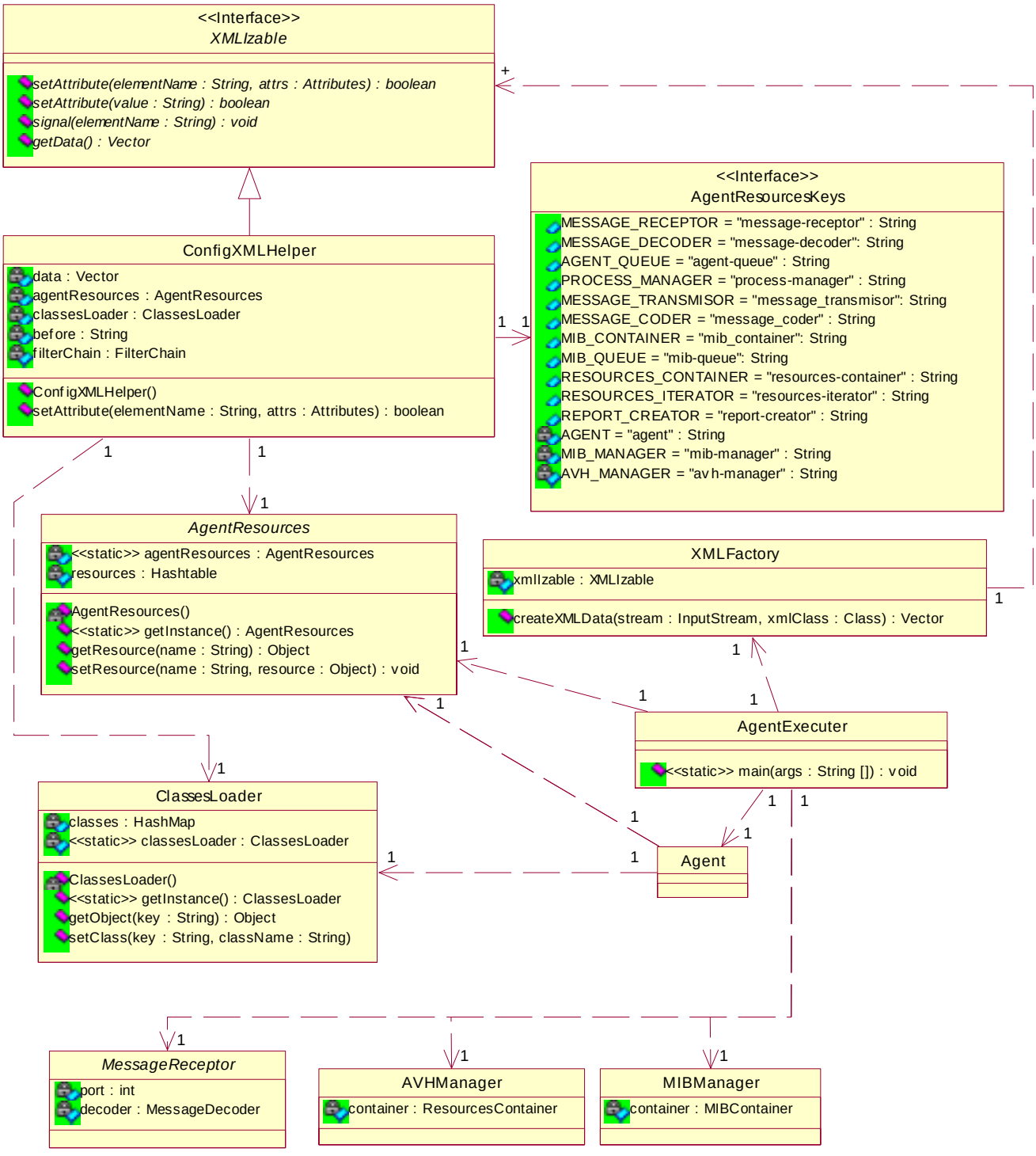


Figura 6.14 Modelo Estático de la Solución de Diseño del "*Configurador del Framework*".

La aplicación de los patrones de diseño se describe a continuación:

- ***Factory Method***. Este patrón de diseño es implementado a través de la clase *XMLFactory* en su método *createXMLData* debido a que la clase de parseo de XML delega el análisis del mismo a través de una clase auxiliar, se decide crear una fábrica de clases auxiliares que permitan analizar cualquier tipo de flujo XML. El *factory method* utiliza también el patrón de diseño *Dynamic Linkage* para crear dinámicamente instancias de clases auxiliares de análisis de XML.

- ***Dynamic Linkage***. Este patrón de diseño es implementado a través de la clase *ConfigXMLHelper*, ya que a través de cadenas obtenidas del archivo configurador del framework se obtienen las clases concretas que definen a la aplicación del Agente. Estas clases concretas se ponen disponibles a través de *ClassesLoader* y algunos elementos importantes se instancian dinámicamente y se ponen disponibles a través de *AgentResources*.

6.8.1.5 Descripción de Clases

La descripción de las clases *AgentExecuter*, *ConfigXMLHelper*, *XMLFactory* y *XMLIzable* se encuentra definida en el "*Anexo C. Descripción de Clases del Framework*" en su "*Sección C.5 Descripción de Clases del Configurador del Framework*".

6.8.1.6 Lógica de Control del Framework

La lógica de control del Configurador del Framework se encuentra principalmente en dos métodos, *setAttribute(…)* de la clase *ConfigXMLHelper* y *main(…)* de la clase *AgentExecuter*.

En el método *setAttribute(…)* se almacenan, en la instancia de *AgentResources*, los Recursos con los cuales contará el agente para su funcionamiento adecuado, como lo son la cola de mensajes del Agente y de la MIB, el contenedor de Objetos Administrados, el iterador de Recursos de la AVH, etc. También en este método se ponen disponibles, a través de una instancia de *ClassesLoader*, objetos Class que representan a clases que heredan de *Service*, esto con el fin de permitir el enlace dinámico al momento en que el Agente obtiene el identificador de servicio de un mensaje y pretenda obtener el servicio correpondiente.

En el método *main(…)* se inicia la ejecución de los cuatro hilos de control principales de la aplicación Agente, el Receptor de Mensajes (Instancia de la clase *MessageReceptor*), el Agente (Instancia de la clase *Agent*), el Administrador de la MIB (Instancia de la clase *MIBManager*) y el Administrador de la AVH (Instancia de la clase *AVHManager*).

Como puede observarse, estos métodos son muy importantes, y no deben ser modificados, ya que son los que permiten la configuración e iniciación de la aplicación del Agente.

6.8.1.7 Extensibilidad

En el presente modelo estático no existe alguna clase de la cual se deba extender, ya que el mecanismo de configuración está definido. No obstante la extensibilidad completa del framework se encuentra en la configuración del archivo XML, en donde se deben declarar las clases que permitan una funcionalidad concreta y completa del Agente.

6.8.1.8 Modelo Dinámico

La Figura 6.15 presentan el modelo dinámico entre instancias de las *AgentExecuter*, *XMLFactory*, *ConfigXMLHelper*, *AgentResources* y *ClassesLoader*, lo que representa la configuración del framework. La Figura 6.16 presenta la interacción entre instancias de las clases *AgentExecuter*, *AgentResources*, *Agent*, *MIBManager* y *AVHManager*, *MessageReceptor*, lo que representa la inicialización de la aplicación Agente.

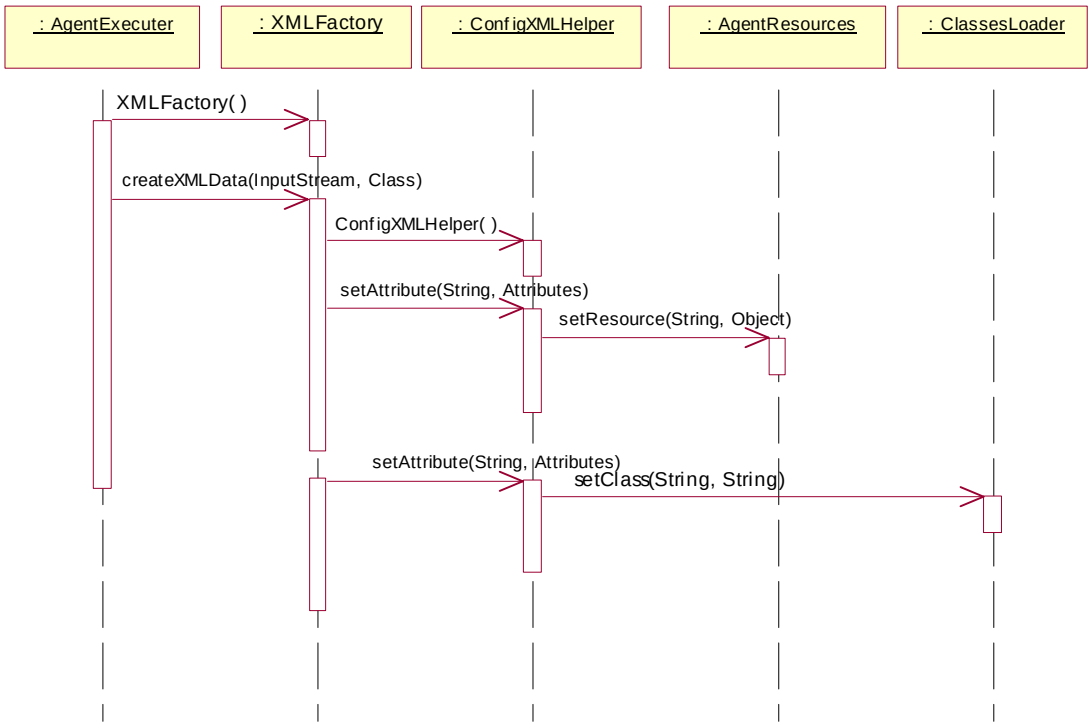


Figura 6.15 Modelo Dinámico de la Solución de Diseño al "*Configurador del Framework*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. *AgentExecuter* inicia la configuración del framework al llamar al método *createXMLData(…)*, el cual crea una instancia de la clase *XMLConfigXMLHelper*, quien obtiene del flujo XML las clases concretas de la aplicación.
2. La lógica de configuración se realiza a través de subsecuentes llamadas al método *setAttribute(…)* de *ConfigXMLHelper*, en donde se pasa como parámetro el nombre de la etiqueta XML a analizar y obtiene los atributos de la misma. Dentro de los atributos se definen las clases concretas de la aplicación, las cuales permiten obtener ya sea objetos *Class* (En el caso de clases que hereden de *Service*), o instancias de las mismas, las cuales se ponen disponibles para su uso en la única instancia de *AgentResources*.

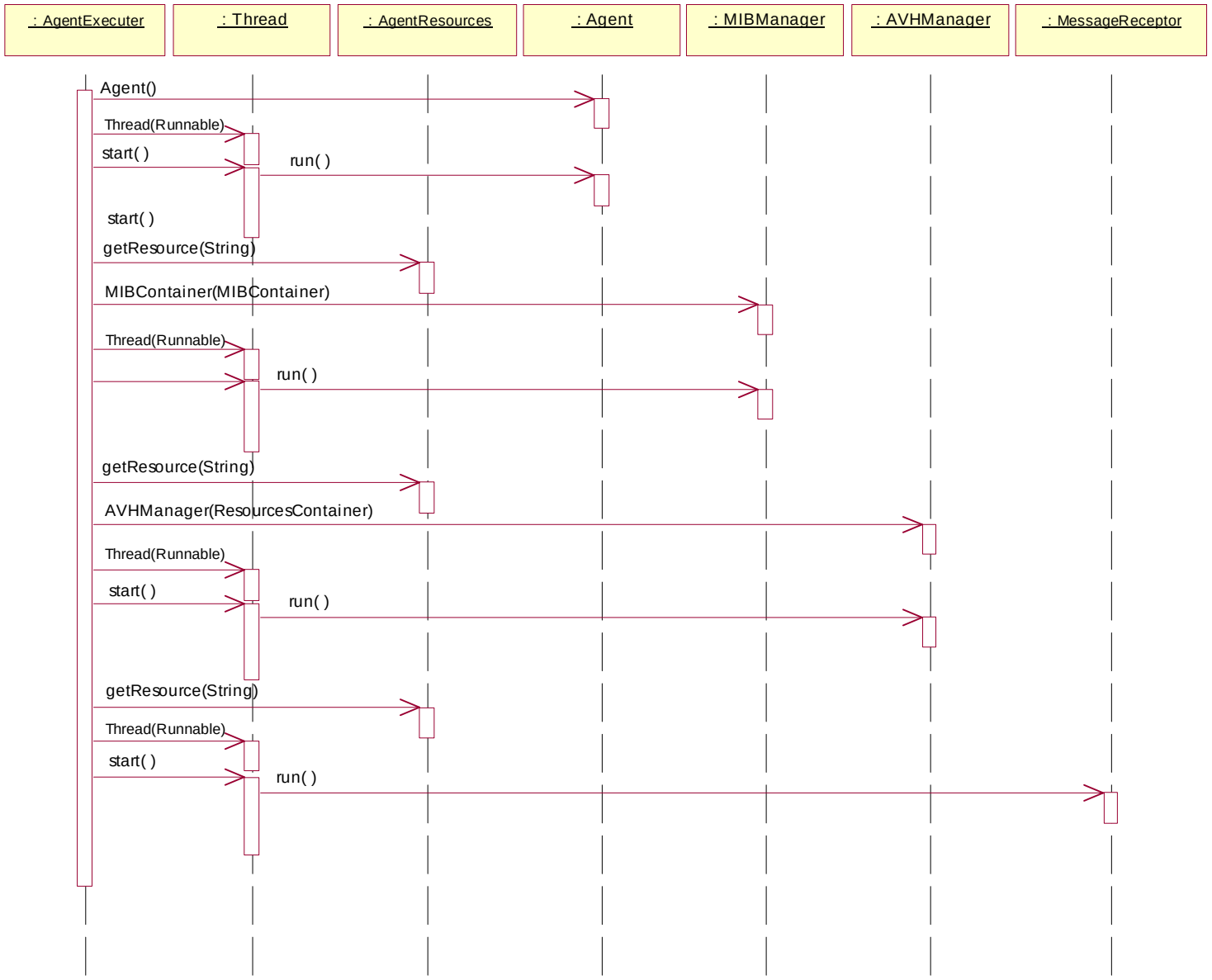


Figura 6.16 Modelo Dinámico de la Solución de Diseño al "*Proceso de Inicialización de la Aplicación Agente*".

El flujo de los mensajes del modelo dinámico es el siguiente:

1. Una vez que *AgentExecuter* terminó la configuración de la aplicación inicia los cuatro hilos de control principales de la aplicación, *Agent*, *MIBManager*, *AVHManager* y *MessageReceptor*, los cuales son obtenidos a través del llamado a la función *getResource(…)* de la instancia de *AgentResources*, pasándole como parámetro la llave que corresponda.

6.9 Modelo Estático de Objetos de Dominio

El Modelo Estático de Objetos de Dominio consiste en la unión de los modelos resultantes de la solución a los conceptos de dominio. En este modelo integral, se pueden apreciar en su totalidad las relaciones de herencia, delegación y dependencia entre las clases del framework. Este modelo se muestra en la Figura 6.17.

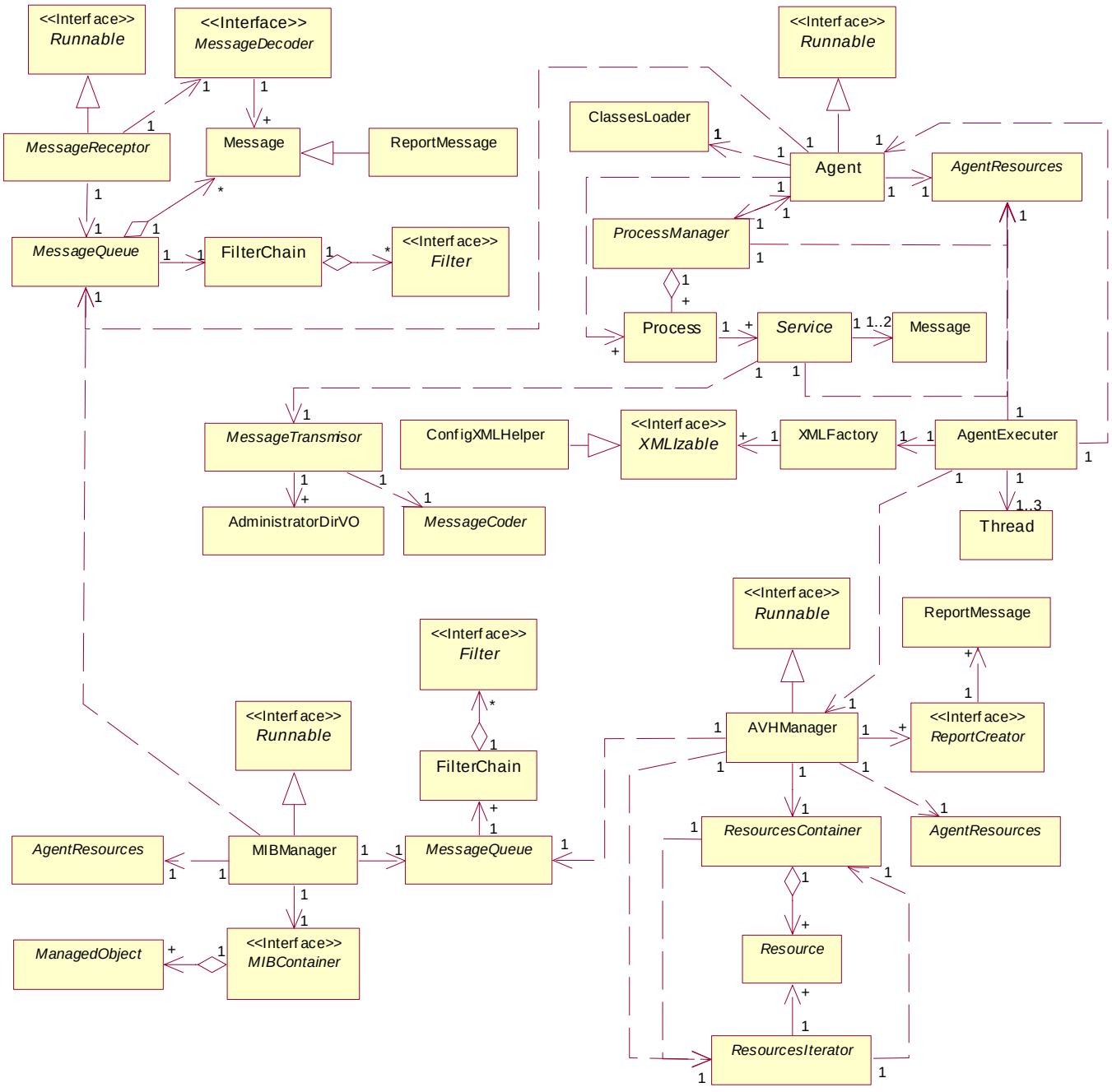


Figura 6.17 Modelo Estático de Objetos de Dominio.

6.10 Modelo Dinámico de Objetos de Dominio

El Modelo Dinámico de Objetos de Dominio consiste en presentar los tres procesos más comunes en la ejecución de una aplicación Agente que utiliza el framewrok. Cabe destacar que la secuencia de mensajes de los tres procesos es la unión de los modelos dinámicos resultantes de la solución a los conceptos del dominio. Los tres procesos son:

1. *Configuración e Inicialización del Framework*. Este proceso permite configurar al framework, lo cual permite definir a una aplicación. El proceso consta de obtener las clases definidas en el archivo XML y convertirlas en instancias correspondientes a recursos de software, las cuales definen la funcionalidad esperada en las capas del Agente. También se inician los principales hilos de control que permiten atender tanto comandos provenientes del Administrador como reportes de cambio de estado de los Recursos. Este proceso se muestra en la Figura 6.18

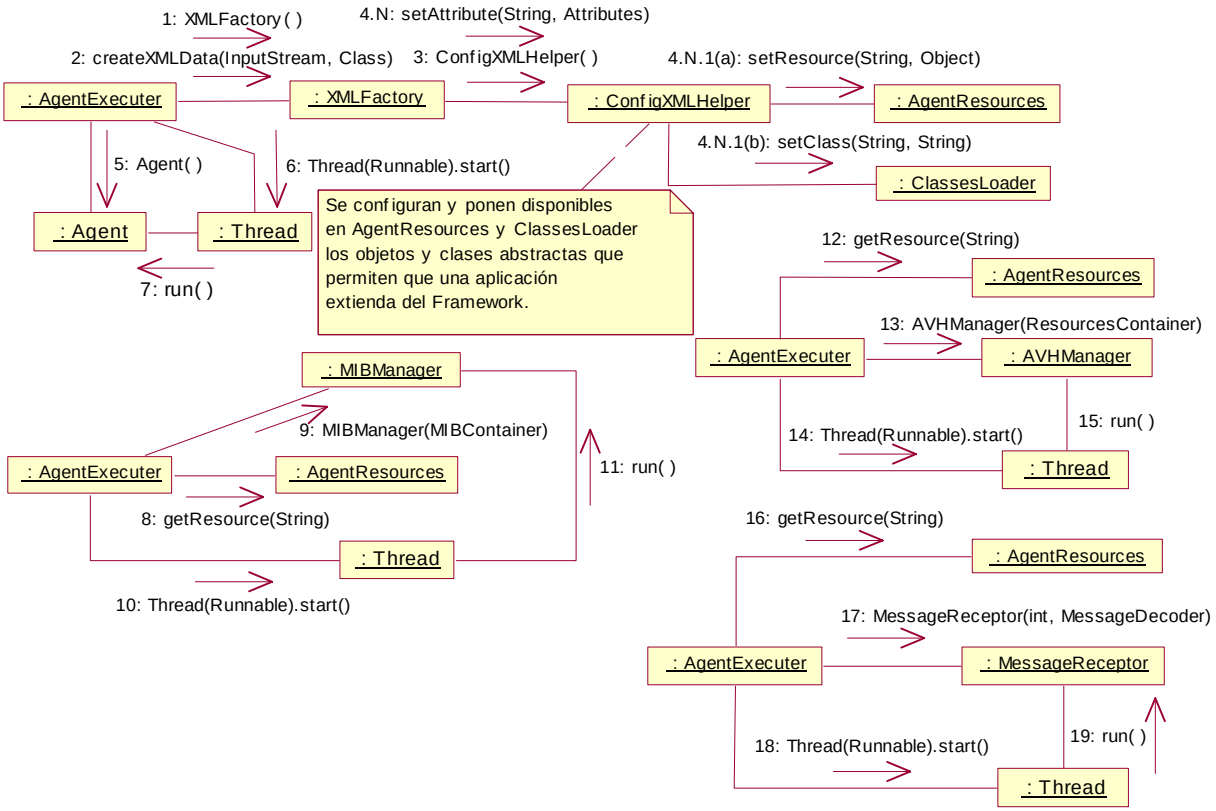


Figura 6.18 Modelo Dinámico de Objetos de Dominio aplicado a la Configuración e Inicialización del Framework.

2. *Interpretación, Filtrado y Ejecución de Comandos provenientes del Administrador*. Este proceso atiende las solicitudes provenientes del Administrador, relacionando el identificador de la solicitud con el Servicio que corresponda, el Servicio puede hacer uso de recursos como el Administrador de la MIB para obtener objetos administrados y realizar acciones en éstos. En caso de que la acción genere un mensaje de retorno, éste es enviado al Administrador. El proceso se muestra en la Figura 6.19.

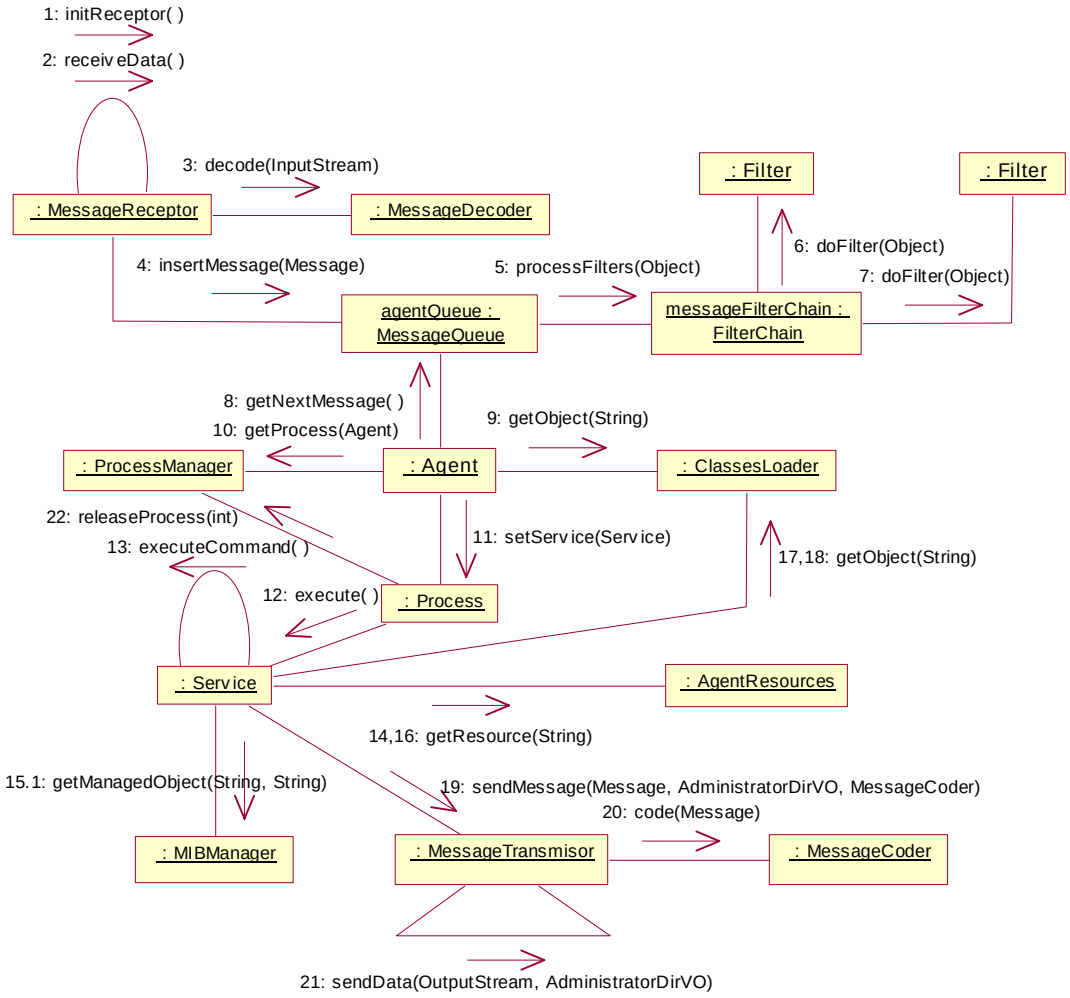


Figura 6.19 Modelo Dinámico de Objetos de Dominio aplicado al Proceso de Interpretación, Filtrado y Ejecución de Comandos que provienen del Administrador.

3. *Envío de Reportes de Cambio de Estado en los Recursos*. Este proceso monitorea los Recursos del elemento de red, y en caso de que uno de ellos cambie de estado, el Administrador de la AVH genera un reporte. El reporte es enviado al Administrador de la MIB, el cual verifica si es necesario que el Agente deba enviarlo al Administrador. En caso afirmativo el reporte es convertido en un flujo de datos y enviado al Administrador. El proceso se muestra en la Figura 6.20.

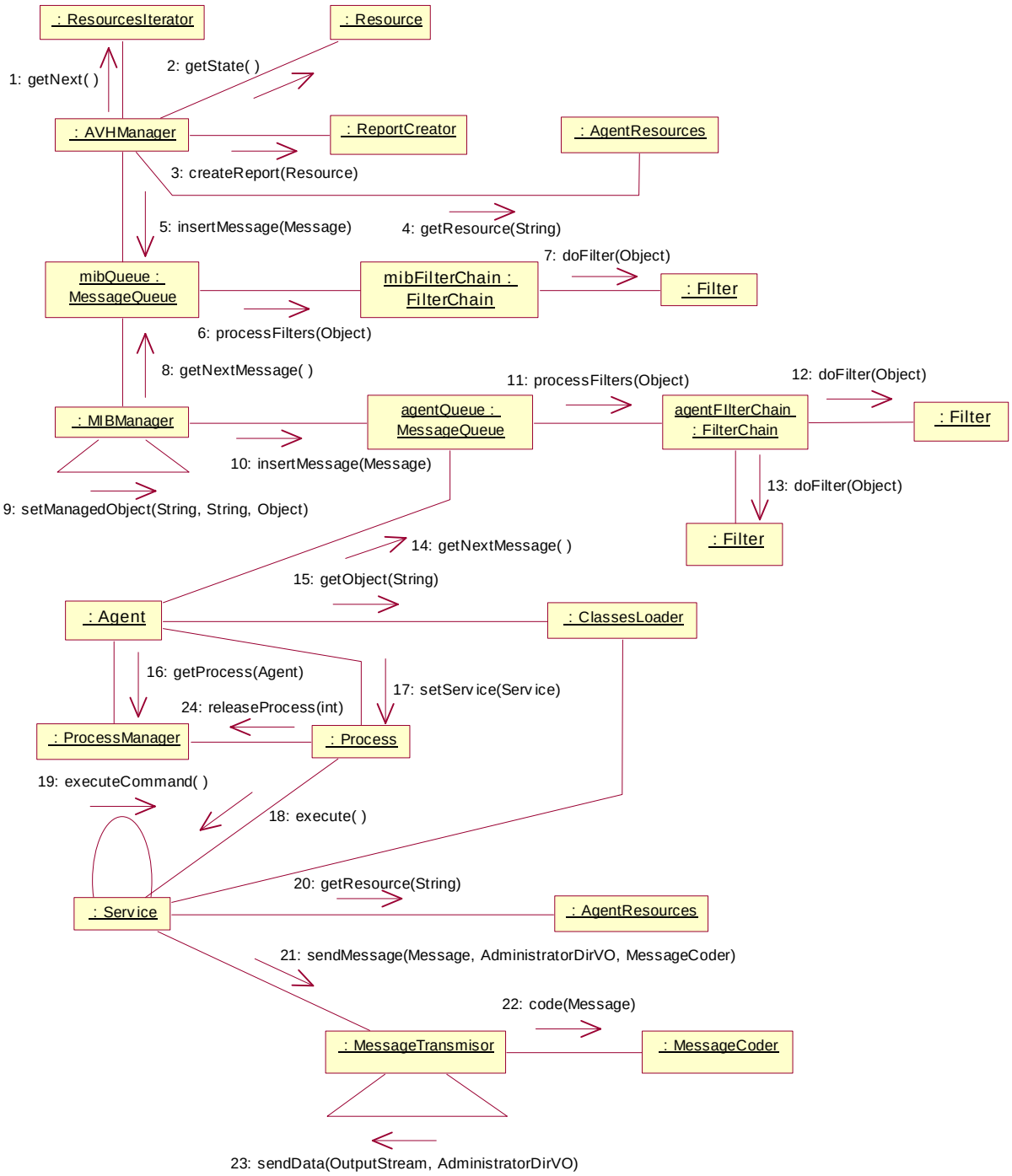


Figura 6.20 Modelo Dinámico de Objetos de Dominio aplicado al Proceso de Envío de Reportes de Cambio de Estado en los Recursos.

6.11 Conclusiones

En el presente capítulo se tienen las siguientes conclusiones:

- El diseño de dominio orientado a objetos aplicado a los Agentes de Administración de Red, permitió diseñar un framework el cual puede ser utilizado para implementar Agentes acorde al modelo OSI o al modelo de Internet, ya que es lo suficientemente flexible como para definir servicios CMIP o SNMP según corresponda.

- El framework mapea los conceptos de dominio determinados en la fase del ADOO y soluciona las problemáticas detectadas en la fase de AVCD.

- El framework es de caja negra, ya que el desarrollador de aplicaciones solo tiene que concretizar las clases que definen la funcionalidad del Agente, sin necesidad de conocer a profundidad el flujo de control del framework.

- La configuración del framework a través de un archivo XML, el cual permite que las aplicaciones sean definidas con mucha facilidad sin necesidad de recompilar las clases del framework, y en caso de que se requiera generar otra aplicación, solamente es necesario redefinir los atributos del archivo.

En el siguiente capítulo se presenta la implementación del framework, el cual está basado en el diseño de dominio.

Capítulo

7

Implementación del Framework de Dominio

7.1 Introducción

En el presente capítulo se presenta la Implementación del Framework de Dominio (IFD), la cual es la etapa de la Ingeniería de Dominio Orientada a Objetos que presenta la codificación de las clases determinadas en el Diseño de Dominio Orientado a Objetos (DDOO).

Las clases son agrupadas en los componentes del dominio determinados en el Análisis de Variabilidad en los Componentes del Dominio (Véase el "*Capítulo 5. Análisis de Variabilidad en los Componentes del Dominio*"). Se presentan también clases auxiliares que facilitan la ejecución de servicios en el framewrok, a las cuales se les denomina *utilerías.*

7.2 Definición

La Implementación del Framework de Dominio es una de las etapas finales de la IDOO, en la cual se reciben como entrada tanto el Modelo Estatico de Objetos de Dominio (MEOD) como el Modelo Dinámico de Objetos de Dominio (MDOD) para crear las clases e instancias determinadas en los mismos.

Para la creación de las clases se utiliza un lenguaje de proramación orientado a objetos el cual pueda soportar en su totalidad el diseño de clases y mecanismos de instanciación, enlace dinámico, etc. definidas en el DDOO.

7.3 Descripción de la Fase de Implementación del Framework de Dominio

La fase de IFD consta de la implementación de las clases definidas en el DDOO. Cada una de éstas clases se agrupa por componentes de dominio, los cuales se describen en el AVCD.

En la fase de IFD aplicada al framework, se determinaron cinco secciones, en las cuales se presentan las clases correspondientes a los componentes del dominio "*Agente*", "*MIB*" y "*AVH*". Además de la sección de "*Utilerías del Framework*" la cual consiste de clases auxiliares en la ejecución de servicios brindados por el Agente, y la sección "*Configurador del Framework*" en donde se implementa el mecanismo de configuración de aplicaciones que utilizan el framework.

En la implementación del framework se decidió codificar las clases utilizando el lenguaje de programación Java, debido a que algunos mecanismos de diseño utilizan el enlace dinámico de clases, lo cual es fácilmente implementado en Java, además de que Java es en la actualidad uno de los lenguajes de programación orientado a objetos de mas uso.

A continuación se presentan las secciones que implementan al framework que permite desarrollar Agentes de Administración de Red.

7.4 Implementación del Componente "Agente"

La implementación del componente o capa del Agente consta de las siguientes clases:

- *AdministratorDirVO*. Representa la dirección IP y puerto hacia donde se deben enviar mensajes al Administrador.
- *Agent*. Hilo de control que interpreta y ejecuta comandos provenientes del Administrador o de reportes de cambio de estado.
- *AgentExecuter*. Inicia la ejecución de las tres capas del elemento de Red (Capas Agente, MIB y AVH), además del receptor de mensajes.
- *Message*. Encapsula información referente a un mensaje.
- *MessageCoder*. Codificador de mensajes que los convierte en flujos de datos de bajo nivel.
- *MessageDecoder*. Decodificador de mensajes que convierte flujos de datos de bajo nivel en mensajes.
- *MessageQueue*. Contenedor de mensajes.
- *MessageReceptor*. Hilo de control que recibe los mensajes enviados por el Administrador.
- *MessageTransmisor*. Permite enviar mensajes producidos por el Agente hacia el Administrador.
- *Process*. Ejecuta servicios solicitados por el Administrador o en su defecto por reportes de cambio de estado.
- *ProcessManager*. Administra los procesos que permiten ejecutar los servicios solicitados.
- *ReportMessage*. Mensaje que representa un reporte de cambios de estado en los Recursos del Elemento de Red o en Objetos Administrados.
- *Service*. Contiene la lógica de negocio que debe llevarse a cabo para ejecutar un comando del Protocolo de Administración de Red utilizado.

En las siguientes secciones se presenta la implementación de las clases anteriores.

7.4.1 AdministratorDirVO

```
/**
 * @(#)com.networkagent.framework.agent.AdministratorDirVO.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  10 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que encapsula información referente a la dirección IP y
* Puerto de Entrada, las cuales son utilizadas para enviar Mensajes
* de Respuesta o Reportes de Cambio de Estado.
*/

public class AdministratorDirVO {
   private String   ip;
   private int      port;

   public AdministratorDirVO(String ip, int port) {
      this.ip   = ip;
      this.port = port;
   }

   public String getIP() {
      return this.ip;
   }

   public int getPort() {
      return this.port;
   }
}
```

7.4.2 Agent

```
/**
 * @(#)com.networkagent.framework.agent.Agent.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Uno de los hilos de control principales del framework.
* Clase que contiene la lógica de negocio de un Agente de Administración de Red
* respecto a la interpretación y ejecución de comandos del Protocolo
* de Administración de Red utilizado.
*/

import com.networkagent.framework.agent.ProcessManager;
import com.networkagent.framework.agent.Process;
import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.Service;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.ClassesLoader;

public class Agent implements Runnable
{
   public void run() {
      AgentResources   agentResources;
      ClassesLoader classesLoader;
      MessageQueue  messageQueue;
      ProcessManager   processManager;

      agentResources   = AgentResources.getInstance();
      classesLoader = ClassesLoader.getInstance();
      messageQueue  = (MessageQueue) agentResources.getResource(
                        AgentResourcesKeys.AGENT_QUEUE);
      processManager   = (ProcessManager) agentResources.getResource(
                        AgentResourcesKeys.PROCESS_MANAGER);

      while (true) {
         Message message = messageQueue.getNextMessage();
       if (message != null) {
       Service service = (Service)classesLoader.getObject(message.getServiceId());
       if (service != null) {
          Process process = processManager.getProcess(this);
          process.setService(service);
          process.notify();
       }
        }
      }
   }
}
```

7.4.3 AgentExecuter

```
/**
 * @(#)com.networkagent.framework.agent.AgentExecuter.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  17 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que ejecuta la aplicación que extiende del framework.
* Utiliza las clases definidas en el archivo XML de configuración.
* Genera los cuatro hilos de control principales, los cuales son:
* + Agent.
* + MIBManager.
* + AVHManager.
* + MessageReceptor.
*/

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Vector;
import java.lang.ClassNotFoundException;

import org.xml.sax.SAXNotRecognizedException;
import org.xml.sax.SAXException;

import com.networkagent.framework.agent.Agent;
import com.networkagent.framework.agent.MessageReceptor;
import com.networkagent.framework.mib.MIBManager;
import com.networkagent.framework.mib.MIBContainer;
import com.networkagent.framework.avh.AVHManager;
import com.networkagent.framework.avh.ResourcesContainer;
import com.networkagent.framework.config.ConfigXMLHelper;
import com.networkagent.framework.config.XMLFactory;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;

public class AgentExecuter
{
   public void main(String [] args) {
   File            file;
   String          xmlFilePath;
   Vector          data;
   FileInputStream fileInputStream;

   XMLFactory    xmlFactory;
   AgentResources   agentResources;
   Agent     agent;
   MIBManager       mibManager;
   AVHManager       avhManager;

   xmlFilePath = args[0];
   if (xmlFilePath == null) {
      System.out.println("Se debe proporcionar la ruta del archivo XML de configuración");
      return;
   }

   xmlFactory    = new XMLFactory();
   file   = new File(xmlFilePath);
   if (file.exists() == false) {
      System.out.println("La ruta del archivo XML de configuración no existe.");
      return;
```

```
   } else {
      try {
         fileInputStream = new FileInputStream(file);
         xmlFactory.createXMLData(fileInputStream,Class.forName(
         "com.networkagent.framework.config.ConfigXMLHelper"));
      } catch (FileNotFoundException fnfexc) {
         System.out.println ("Error al intentar cargar el archivo de configuración.
                              Excepción: " + fnfexc.toString());
         return;
      } catch (IOException ioexc) {
         System.out.println ("Error al intentar cargar el archivo de configuración.
         Excepción: " + ioexc.toString());
         return;
      } catch (ClassNotFoundException cnfexc) {
         System.out.println ("Error al intentar obtener la clase de configuración del
                              archivo. Excepción: " + cnfexc.toString());
         return;
      } catch (SAXNotRecognizedException snrexc) {
         System.out.println ("Error al intentar obtener la clase de configuración del
                           archivo. Excepción: " + snrexc.toString());
         return;
      } catch (SAXException saxexc) {
         System.out.println ("Error al intentar obtener la clase de configuración del
                              archivo. Excepción: " + saxexc.toString());
         return;
      }
   }

   agentResources  = AgentResources.getInstance();
   agent           = new Agent();
   mibManager       = new MIBManager((MIBContainer) agentResources.getResource(
                      AgentResourcesKeys.MIB_CONTAINER));
   avhManager      = new AVHManager((ResourcesContainer) agentResources.getResource(
                      AgentResourcesKeys.RESOURCES_CONTAINER));

   agentResources.setResource(AgentResourcesKeys.AGENT, agent);
   agentResources.setResource(AgentResourcesKeys.MIB_MANAGER, mibManager);
   agentResources.setResource(AgentResourcesKeys.AVH_MANAGER, avhManager);

   new Thread(agent).start();
   new Thread(mibManager).start();
   new Thread(avhManager).start();
   new Thread((MessageReceptor) agentResources.getResource(
              AgentResourcesKeys.MESSAGE_RECEPTOR)).start();

   }
}
```

7.4.4 Message

```
/**
 * @(#)com.networkagent.framework.agent.Message.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  10 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que encapsula información relevante de un Mensaje, como su
* identificador, así como el identificador del servicio con el cual
* se relaciona.
*/

public class Message
{
   private String serviceId;
   private String messageId;

   public String getServiceId() {
      return this.serviceId;
   }

   public String getMessageId() {
      return this.messageId;
   }

   public void setServiceId(String serviceId) {
      this.serviceId = serviceId;
   }

   public void setMessageId(String messageId) {
      this.messageId = messageId;
   }
}
```

7.4.5 MessageCoder

```
/**
 * @(#)com.networkagent.framework.agent.MessageCoder.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Interfaz en la cual se define el método que permite codificar mensajes
* convirtiéndolos en un flujo de salida.
*/

import java.io.OutputStream;

import com.networkagent.framework.agent.Message;

public interface MessageCoder {
   public abstract OutputStream code(Message message);
}
```

7.4.6 MessageDecoder

```
/**
 * @(#)com.networkagent.framework.agent.MessageDecoder.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Interfaz en la cual se define el método que permite decodificar flujos
* de entrada en Mensajes.
*/

import java.io.InputStream;

import com.networkagent.framework.agent.Message;

public interface MessageDecoder {
   public abstract Message decode(InputStream stream);
}
```

7.4.7 MessageQueue

```
/**
 * @(#)com.networkagent.framework.agent.MessageQueue.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  12 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que permite almacenar mensajes provenientes del Administrador
* o del Administrador Virtual de Hardware.
*/

import java.util.Vector;

import com.networkagent.framework.agent.Message;
import com.networkagent.framework.util.FilterChain;

public abstract class MessageQueue
{
   private Vector      messagesList;
   private int      maxSize;
   private FilterChain queueFilters;

   public MessageQueue(int maxSize, FilterChain filterChain) {
      this.messagesList= new Vector();
      this.maxSize     = maxSize;
      this.queueFilters= queueFilters;
   }

   public void insertMessage(Message message) {
      message = (Message) this.queueFilters.processFilters(message);

      if (message != null) {
         if (this.messagesList.size() < this.maxSize) {
            this.messagesList.add(message);
         }
      }
   }

   public abstract Message getNextMessage();

   public int getMaxSize() {
      return this.maxSize;
   }

   public int getSize() {
      return this.messagesList.size();
   }
}
```

7.4.8 MessageReceptor

```
/**
 * @(#)com.networkagent.framework.agent.MessageReceptor.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Uno de los hilos de control principales del framework.
* Permite recibir los mensajes provenientes del Administrador.
*/

import java.lang.Runnable;
import java.io.InputStream;

import com.networkagent.framework.agent.MessageDecoder;
import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.AgentResources;

public abstract class MessageReceptor implements Runnable {
   private int    port;
   private MessageDecoder  decoder;

   MessageReceptor(int port, MessageDecoder decoder) {
      this.port    = port;
      this.decoder = decoder;
   }

   public void run() {
      AgentResources   agentResources;
      MessageQueue  messageQueue;
      Message    message;
      InputStream   inputStream;

      agentResources   = AgentResources.getInstance();
      messageQueue  = (MessageQueue) agentResources.getResource(
                        AgentResourcesKeys.AGENT_QUEUE);
      this.initReceptor();

      while(true) {
         inputStream  = this.receiveData();
         message      = this.decoder.decode(inputStream);
         messageQueue.insertMessage(message);
      }
   }

   protected abstract void initReceptor();
   protected abstract void destroyReceptor();
   protected abstract InputStream receiveData();
}
```

7.4.9 MessageTransmisor

```
/**
 * @(#)com.networkagent.framework.agent.MessageTransmisor.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  14 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que permite enviar al Administrador mensajes de respuesta
* de ejecución de comandos o reportes de cambio de estado.
*/

import java.io.OutputStream;

import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.MessageCoder;
import com.networkagent.framework.agent.AdministratorDirVO;

public abstract class MessageTransmisor {

   public void sendMessage(Message message, AdministratorDirVO administratorDir,
                           MessageCoder coder) {
      OutputStream stream;

      stream = coder.code(message);
      if (stream != null) {
     this.sendData(stream, administratorDir);
      }
   }

   protected abstract void sendData(OutputStream stream,
                                    AdministratorDirVO administratorDir);
}
```

7.4.10 Process

```
/**
 * @(#)com.networkagent.framework.agent.Process.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  12 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Hilo de control que permite ejecutar servicios solicitados por el
* Administrador o servicios de envío de mensajes. Este hilo de control
* pertenece a el pool definido en ProcessManager.
*/


import java.lang.Runnable;
import java.lang.InterruptedException;

import com.networkagent.framework.agent.Service;
import com.networkagent.framework.agent.ProcessManager;

public class Process implements Runnable {
   private Service     service;
   private ProcessManager  processManager;
   private int         processId;

   public Process(ProcessManager processManager, int processId) {
      this.processManager = processManager;
      this.processId      = processId;
   }

   public void run() {

      while(true) {
     if (this.service != null) {
        this.service.execute();
        this.service = null;
         }

         this.processManager.releaseProcess(this.processId);
         try {
        this.wait();
         } catch(InterruptedException iexc) {
            System.out.println ("Error al intentar para el Proceso #" + this.processId +
                                ". Exception: " + iexc.toString());
         }
      }
   }

   public int getProcessId() {
      return this.processId;
   }

   public Service getService() {
      return this.service;
   }

   public void setService(Service service) {
      this.service = service;
   }
}
```

7.4.11 ProcessManager

```
/**
 * @(#)com.networkagent.framework.agent.Process.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  13 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Administrador de Procesos, el cual contiene un pool que contiene
* hilos de control para la ejecución de comandos de protocolo de admón.
* de red.
*/

import java.util.Vector;

import com.networkagent.framework.agent.Process;
import com.networkagent.framework.agent.Agent;

public abstract class ProcessManager {
   private Vector   processList;
   private int      size;
   private int      busyProcess;

   public ProcessManager(int size) {
      this.processList = new Vector();

      for(int i=0; i < size; i++) {
         Process process = new Process(this, i);
         this.processList.add(process);
      }
   }

   public int getSize() {
      return this.size;
   }

   public Process getProcess(Agent agent) {
      while(this.size <= this.busyProcess);

      this.busyProcess ++;
      return this.getProcess();
   }

   protected abstract Process getProcess();
   public abstract void releaseProcess(int processId);
   public abstract void suspendProcess(int processId);
}
```

7.4.12 ReportMessage

```
/**
 * @(#)com.networkagent.framework.agent.ReportMessage.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  14 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que encapsula información relevante de un Mensaje que representa
* un cambio de estado en Recursos del Elemento de Red.
*/
import com.networkagent.framework.agent.Message;

public class ReportMessage extends Message
{
   private String objectId;
   private String instance;
   private Object value;

   public ReportMessage(String objectId, String instance, Object value) {
      this.objectId = objectId;
      this.instance = instance;
      this.value    = value;
   }

   public String getObjectId() {
      return this.objectId;
   }

   public String getInstance() {
      return this.instance;
   }

   public Object getValue() {
    return this.value;
   }
}
```

7.4.13 Service

```
/**
 * @(#)com.networkagent.framework.agent.Service.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  14 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.agent;

/**
* Clase que encapsula la lógica de negocio de un Servicio que puede
* brindar el Agente, el Servicio representa la ejecución de un comando
* del protocolo de Administración de Red.
*/

import java.util.Vector;
import java.util.Iterator;

import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.MessageTransmisor;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.ClassesLoader;


public abstract class Service
{
   private String   serviceId;
   private Message messageIn;
   private Message messageOut;

   public Service(String serviceId, Message message) {
      this.serviceId = serviceId;
      this.messageIn = message;
   }

   public void execute() {
      AgentResources      resources;
      ClassesLoader       loader;
      MessageTransmisor   transmisor;
      MessageCoder        coder;
      Vector        administrators;
      Iterator         iterator;

      this.messageOut = this.executeCommand();
      resources       = AgentResources.getInstance();
      loader          = ClassesLoader.getInstance();

      if (this.messageOut != null) {
         transmisor     = (MessageTransmisor) loader.getObject(
                           AgentResourcesKeys.MESSAGE_TRANSMISOR);
         coder          = (MessageCoder) loader.getObject(
                           AgentResourcesKeys.MESSAGE_CODER);
         administrators = (Vector) resources.getResource(
                           AgentResourcesKeys.ADMINISTRATORS);
         iterator       = administrators.iterator();

         while(iterator.hasNext()) {
       AdministratorDirVO administratorDir;

       administratorDir = (AdministratorDirVO)iterator.next();
       transmisor.sendMessage(this.messageOut, administratorDir, coder);
         }
      }
```

```
   }

   protected abstract Message executeCommand();

   public String getServiceId() {
      return this.serviceId;
   }

   public Message getMessageIn() {
      return this.messageIn;
   }

   public Message getMesageOut() {
      return this.messageOut;
   }
}
```

7.5 Implementación del Componente "MIB"

La implementación de la MIB consta de las siguientes clases:

- *MIBContainer*. Clase contenedora de los Objetos Administrados que pertenecen a la MIB.
- *MIBManager*. Administrador de la MIB, el cual es un proxy que permite obtener los Objetos Administrados que se soliciten, en representación del *MIBContainer*. También revisa los reportes de cambio de estado y en su caso los envía al Agente para su atención.
- *ManagedObject*. Representa a una clase de Objeto Administrado, encapsulando su información.

7.5.1 MIBContainer

```
/**
 * @(#)com.networkagent.framework.mib.MIBContainer.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.mib;

/**
* Interfaz que define los métodos que deben ser implementados
* por el contenedor de Objetos Administrados de la MIB.
*/

public interface MIBContainer {
   public abstract void initialize();
   public abstract ManagedObject getManagedObject(String managedObjectId, String instance);
   public abstract ManagedObject insertManagedObject(ManagedObject managedObject);
   public abstract ManagedObject deleteManagedObject(String managedObjectId, String instance);
}
```

7.5.2 MIBContainer

```
/**
 * @(#)com.networkagent.framework.mib.MIBManager.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.mib;

/**
* Uno de los hilos de control principales del framework.
* Clase que contiene la lógica de negocio de la Base de Información
* de Administración, la cual se encarga de atender mensajes de
* reportes de cambio de estado en los Recursos del Elemento de Red,
* los cuales a su vez son enviados al Agente para que los atienda.
*/

import java.lang.Runnable;

import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.ReportMessage;
import com.networkagent.framework.mib.MIBContainer;

import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;

public class MIBManager implements Runnable
{
   private   MIBContainer container;

   public MIBManager(MIBContainer container) {
      this.container = container;
   }

   public void run() {
      AgentResources   agentResources;
      MessageQueue  mibQueue;
      MessageQueue  agentQueue;

      agentResources   = AgentResources.getInstance();
      mibQueue   = (MessageQueue) agentResources.getResource(
                        AgentResourcesKeys.MIB_QUEUE);
      agentQueue = (MessageQueue) agentResources.getResource(
                        AgentResourcesKeys.AGENT_QUEUE);

      while(true) {
         ReportMessage reportMessage;

         reportMessage = (ReportMessage) mibQueue.getNextMessage();

         if (reportMessage != null) {
            ManagedObject managedObject;

            managedObject = this.setManagedObject(reportMessage.getObjectId(),
                           reportMessage.getInstance(), reportMessage.getValue());

            if (managedObject != null) {
               agentQueue.insertMessage(reportMessage);
            }
         }
      }
   }
```

```
   public ManagedObject getManagedObject(String managedObjectId, String instance) {
      ManagedObject managedObject;

      managedObject = this.container.getManagedObject(managedObjectId, instance);

      if (managedObject != null) {
         if (managedObject.getRead() == false) {
            managedObject = null;
         }
      }

      return managedObject;
   }

   public ManagedObject setManagedObject(String managedObjectId, String instance,
                                         Object value) {
      ManagedObject managedObject;

      managedObject = this.container.getManagedObject(managedObjectId, instance);

      if (managedObject != null) {
         if (managedObject.getWrite() == true) {
            managedObject.setValue(value);
         } else {
            managedObject = null;
         }
      }

      return managedObject;
   }

   public ManagedObject insertManagedObject(ManagedObject managedObject) {
      return this.container.insertManagedObject(managedObject);
   }

   public ManagedObject deleteManagedObject(String managedObjectId, String instance) {
      return this.container.deleteManagedObject(managedObjectId, instance);
   }
}
```

7.5.3 ManagedObject

```
/**
 * @(#)com.networkagent.framework.mib.ManagedObject.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.mib;

/**
* Clase que encapsula información del estado de un Objeto Administrado
*/

public abstract class ManagedObject
{
   private String id;
   private String instance;
   private boolean read;
   private boolean write;

   public ManagedObject(String id, String instance, boolean read, boolean write) {
      this.id = id;
   }

   public String getId() {
      return this.id;
   }

   public String getInstance() {
      return this.instance;
   }

   public boolean getRead() {
      return this.read;
   }

   public boolean getWrite() {
      return this.write;
   }

   public abstract void setValue(Object value);
   public abstract Object getValue();
   public abstract boolean getState();
}
```

7.6 Implementación del Componente "AVH"

La implementación del componente AVH consta de las siguientes clases:

- *AVHManager*. Administrador Virtual de Hardware, el cual realiza monitoreos constantes sobre los Recursos del Elemento de Red.
- *ReportCreator*. Permite crear reportes de cambio de estado en los Recursos.
- *Resource*. Encapsula información relevante de un Recurso de Hardware, además de permitir obtener el estado del mismo.
- *ResoucesContainer*. Contenedor de Recursos.
- *ResourcesIterator*. Iterador de Recursos, el cual es utilizado por el *AVHManager* para monitorear los Recursos.

7.6.1 AVHManager

```
/**
 * @(#)com.networkagent.framework.avh.AVHManager.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.avh;

/**
* Uno de los hilos de control principales del framework.
* Clase que contiene la lógica de negocio del Administrador Virtual
* de Hardware, el cual es el encargado de monitorear los Recursos de
* un Elemento de Red.
*/

import java.lang.Runnable;
import java.lang.Thread;
import java.lang.InterruptedException;

import com.networkagent.framework.agent.ReportMessage;
import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.avh.ReportCreator;
import com.networkagent.framework.avh.ResourcesIterator;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.ClassesLoader;

public class AVHManager implements Runnable
{
   private ResourcesContainer container;
   private long     sleepTime;

   public AVHManager(ResourcesContainer container, long sleepTime) {
      this.container = container;
      this.sleepTime = sleepTime;
   }

   public void run() {
      AgentResources    agentResources;
      ClassesLoader     classesLoader;
      MessageQueue      mibQueue;
      ReportCreator     reportCreator;
      int               sleepTime;
      ResourcesIterator iterator;

      agentResources = AgentResources.getInstance();
      classesLoader  = ClassesLoader.getInstance();
      mibQueue       = (MessageQueue)  agentResources.getResource(
                        AgentResourcesKeys.MIB_QUEUE);
      reportCreator  = (ReportCreator) classesLoader.getObject(
                        AgentResourcesKeys.REPORT_CREATOR);
      iterator       = this.container.getResourcesIterator();

      while(true) {
         Resource resource;
         boolean  resourceState;

         resource    = iterator.getNext();
         resourceState = resource.getState();
```

```
         if (resourceState == false) {
            ReportMessage reportMessage;

            reportMessage = reportCreator.createReport(resource);
            mibQueue.insertMessage(reportMessage);
         }

         try {
            Thread.sleep(this.sleepTime);
         } catch (InterruptedException intexc) {
            System.out.println ("Error al intentar dormir al Thread del AVHManager. Excepción: " +
                                intexc.toString());
         }
      }
   }

   public ResourcesContainer getResourcesContainer() {
      return this.container;
   }

   public ReportMessage createMessageReport(Resource resource) {
      AgentResources   agentResources;
      ReportCreator reportCreator;

      agentResources   = AgentResources.getInstance();
      reportCreator = (ReportCreator) agentResources.getResource(
                        AgentResourcesKeys.REPORT_CREATOR);

      return reportCreator.createReport(resource);
   }
}
```

7.6.2 ReportCreator

```
/**
 * @(#)com.networkagent.framework.avh.ReportCreator.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.avh;

/**
* Clase que permite crear un reporte de cambio de estado en los Recursos
* de un Elemento de Red.
*/

import com.networkagent.framework.agent.ReportMessage;

public interface ReportCreator
{
   public abstract ReportMessage createReport(Resource resource);
}
```

7.6.3 Resource

```
/**
 * @(#)com.networkagent.framework.avh.Resource.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  16 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.avh;

/**
* Clase que encapsula información del estado de un Recurso del Elemento
* de Red.
*/

public abstract class Resource
{
   private String id;
   private String managedObjectId;
   private String managedObjectInstance;

   public Resource(String id, String managedObjectId, String managedObjectInstance) {
      this.id                   = id;
      this.managedObjectId      = managedObjectId;
      this.managedObjectInstance = managedObjectInstance;
   }

   public String getId() {
      return this.id;
   }

   public String getManagedObjectId() {
      return this.managedObjectId;
   }

   public String getManagedObjectInstance() {
      return this.managedObjectInstance;
   }

   public abstract void setValue(Object value);
   public abstract boolean getState();
   public abstract Object getValue();
}
```

7.6.4 ResourcesContainer

```
/**
 * @(#)com.networkagent.framework.avh.ResourcesContainer.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  16 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.avh;

/**
* Clase que representa al contenedor de recursos, los cuales son monitoreados
* a traves del Administrador Virtual de Hardware.
*/

import java.util.Vector;
import java.util.HashMap;
import java.lang.Class;
import java.lang.IllegalAccessException;
import java.lang.InstantiationException;
import java.lang.IllegalArgumentException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Constructor;

import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.ClassesLoader;
import com.networkagent.framework.avh.Resource;
import com.networkagent.framework.avh.ResourcesIterator;

public abstract class ResourcesContainer
{
   private Vector   resourcesList;
   private HashMap resourcesMapping;
   private HashMap resourcesMIBMapping;

   public ResourcesContainer() {
      this.resourcesList    = new Vector();
      this.resourcesMapping    = new HashMap();
      this.resourcesMIBMapping = new HashMap();
      this.initialize();
   }

   public abstract void initialize();

   public Resource getResource(String id) {
      return (Resource) this.resourcesMapping.get(id);
   }

   public Resource getResource(String managedObjectId, String managedObjectInstance) {
      return (Resource) this.resourcesMIBMapping.get(managedObjectId + "_" +
                                                     managedObjectInstance);
   }

   public void insertResource(Resource resource) {
      this.resourcesList.add(resource);
      this.resourcesMapping.put(resource.getId(), resource);
      this.resourcesMIBMapping.put(resource.getManagedObjectId() + "_" +
                                   resource.getManagedObjectInstance(), resource);
   }
```

```
   public Resource deleteResource(String id) {
      Resource resource;

      resource = (Resource) this.resourcesMapping.get(id);

      if (resource != null) {
         this.resourcesMapping.remove(id);
         this.resourcesMIBMapping.remove(resource.getManagedObjectId() + "_" +
                                         resource.getManagedObjectInstance());
         this.resourcesList.remove(resource);
      }

      return resource;
   }

   public ResourcesIterator getResourcesIterator() {
      ClassesLoader      classesLoader;
      Class              iteratorClass;
      Constructor        iteratorConstructor;
      Object[]           iteratorParameters;
      ResourcesIterator  resourcesIterator;

      ResourcesIterator     = null;
      ClassesLoader         = ClassesLoader.getInstance();
      IteratorClass         = classesLoader.getClass(AgentResourcesKeys.RESOURCES_ITERATOR);
      IteratorConstructor   = iteratorClass.getConstructors()[0];
      iteratorParameters    = new Object[1];
      iteratorParameters[0] = this.resourcesList;

      try {
         resourcesIterator = (ResourcesIterator) iteratorConstructor.newInstance(
                                                    iteratorParameters);
      } catch (IllegalAccessException illexc) {
         System.out.println ("Error al intentar crear una instancia de Resources Iterator" +
                             illexc.toString());
      } catch (InstantiationException instexc) {
         System.out.println ("Error al intentar crear una instancia de Resources Iterator" +
                             instexc.toString());
      } catch (IllegalArgumentException illargexc) {
         System.out.println ("Error al intentar crear una instancia de Resources Iterator" +
                             illargexc.toString());
      } catch (InvocationTargetException invexc) {
         System.out.println ("Error al intentar crear una instancia de Resources Iterator" +
                             invexc.toString());
      }

      return resourcesIterator;
   }
}
```

7.6.5 ResourcesIterator

```
/**
 * @(#)com.networkagent.framework.avh.ResourcesIterator.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  16 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.avh;

/**
* Clase que representa al iterador de Recursos del Elemento de Red.
* el cual es utilizado por el AVHManager para obtener Recursos que
* sobre los cuales se verificará el estado de los mismos.
*/

import java.util.Vector;

import com.networkagent.framework.avh.Resource;

public abstract class ResourcesIterator
{
   private Vector resourcesList;

   public ResourcesIterator(Vector resourcesList) {
      this.resourcesList = resourcesList;
   }

   public abstract Resource getNext();
}
```

7.7 Implementación de las Utilerías del Framework

Las utilerías del framework consisten en clases que pueden ser utilizadas en cualquier capa del framework. Este tipo de clases son:

- *AgentResources*. Clase que permite obtener Recursos de Software que auxilien en la funcionalidad o lógica de negocio de los servicios ofrecidos por el Agente en cualquiera de sus capas.
- *AgentResourcesKeys*. Llaves que permiten obtener Recursos de Software contenidos en *AgentResources*.
- *ClassesLoader*. Contenedor dinámico de clases, el cual dada una llave permite obtener una instancia de la clase solicitada.
- *Filter*. Clase que permite filtrar o ejecutar servicios en un Objeto.
- *FilterChain*. Cadena de filtros que se deben aplicar sobre un Objeto.

7.7.1 AgentResources

```
/**
 * @(#)com.networkagent.framework.util.AgentResources.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.util;

/**
* Clase que contiene los Recursos de Software que pueden ser utilizados
* por elementos del Agente de Administración de Red.
*/

import java.util.HashMap;

public class AgentResources {
   private static   AgentResources   agentResources;
   private       HashMap      resources;

   private AgentResources() {
      agentResources.resources = new HashMap();
   }

   public static AgentResources getInstance() {
      if (agentResources == null) {
         agentResources = new AgentResources();
      }

      return agentResources;
   }

   public Object getResource(String name) {
      return agentResources.resources.get(name);
   }

   public void setResource(String name, Object resource) {
      agentResources.resources.put(name, resource);
   }
}
```

7.7.2 AgentResourcesKeys

```
/**
 * @(#)com.networkagent.framework.util.AgentResourcesKeys.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.util;

/**
* Interfaz que contiene las llaves que permiten obtener los recursos de software
* del Agente de Administración de Red almacenadas en AgentResources.
*/

public interface AgentResourcesKeys {
   public static final String ADD_CLASS           = "add-class";
   public static final String ADMINISTRATOR    = "administrator";
   public static final String ADMINISTRATORS      = "administrators";
   public static final String AGENT               = "agent";
   public static final String AGENT_FILTERCHAIN   = "agent-filter-chain";
   public static final String AGENT_QUEUE         = "agent-queue";
   public static final String AVH_MANAGER         = "avh-manager";
   public static final String AVH_CONTAINER    = "avh-container";
   public static final String FILTER              = "filter";
   public static final String MESSAGE_CODER    = "message-coder";
   public static final String MESSAGE_DECODER     = "message-decoder";
   public static final String MESSAGE_RECEPTOR    = "message-receptor";
   public static final String MESSAGE_TRANSMISOR  = "message-transmisor";
   public static final String MIB_CONTAINER    = "mib-container";
   public static final String MIB_FILTERCHAIN     = "mib-filter-chain";
   public static final String MIB_MANAGER         = "mib-manager";
   public static final String MIB_QUEUE           = "mib-queue";
   public static final String PROCESS_MANAGER     = "process-manager";
   public static final String REPORT_CREATOR      = "report-creator";
   public static final String RESOURCE            = "resource";
   public static final String RESOURCES_CONTAINER = "resources-container";
   public static final String RESOURCES_ITERATOR  = "resources-iterator";
   public static final String SERVICE             = "service";
   public static final String SLEEP_TIME          = "sleep-time";
}
```

7.7.3 ClassesLoader

```
/**
 * @(#)com.networkagent.framework.util.ClassesLoader.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.util;

/**
* Clase que contiene Clases auxiliares que son utilizadas por el Agente
* las cuales pueden representar Servicios u otros elementos importantes.
*/

import java.util.HashMap;
import java.lang.Class;
import java.lang.InstantiationException;
import java.lang.IllegalAccessException;
import java.lang.ClassNotFoundException;

public class ClassesLoader
{
   private      HashMap      classes;
   private static   ClassesLoader    classesLoader;

   private ClassesLoader() {
      classesLoader.classes = new HashMap();
   }

   public static ClassesLoader getInstance() {
      if (classesLoader == null) {
         classesLoader = new ClassesLoader();
      }

      return classesLoader;
   }

   public Object getObject(String key) {
      Class  classObj;
      Object object;

      classObj   = (Class)classesLoader.classes.get(key);
      try {
         object  = classObj.newInstance();
      } catch(InstantiationException instexc) {
         System.out.println ("Error al intentar obtener un objeto con la llave:'" + key +
                             "'. Excepción: " + instexc.toString());
         object = null;
      } catch(IllegalAccessException illexc) {
         System.out.println ("Error al intentar obtener un objeto con la llave:'" + key +
                             "'. Excepción: " + illexc.toString());
         object = null;
      }

      return object;
   }

   public Class getClass(String key) {
      Class classObj;

      classObj = (Class)classesLoader.classes.get(key);
      return classObj;
   }
```

```
   public void setClass(String key, String className) {
      Class classObj;

      try {
         classObj = Class.forName(className);
         classesLoader.classes.put(key, classObj);
      } catch(ClassNotFoundException cnfexc) {
         System.out.println ("Error al intentar obtener una clase por su nombre:'" +
                             className + "'. Excepción: " + cnfexc.toString());
      }
   }
}
```

7.7.4 Filter

```
/**
 * @(#)com.networkagent.framework.util.Filter.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.util;

/**
* Interfaz que contiene el método que debe ser implementado para
* crear un filtro que debe ser aplicado sobre los Mensajes o sobre
* cualquier otro elemento que sea útil en el funcionamiento del Agente.
*/

public interface Filter
{
   public Object doFilter(Object object);
}
```

7.7.5 FilterChain

```
/**
 * @(#)com.networkagent.framework.util.FilterChain.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  11 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.util;

/**
* Clase que contiene los filtros que se deben aplicar sobre los Mensajes
* o sobre cualquier elemento importante en el funcionamiento de un
* Agente.
*/

import java.util.Vector;
import java.util.Iterator;

import com.networkagent.framework.util.Filter;

public class FilterChain {
   private Vector filters;

   public FilterChain() {
      this.filters = new Vector();
   }

   public void addFilter(Filter filter) {
      this.filters.add(filter);
   }

   public Object processFilters(Object object) {
      Iterator iterator;

      iterator = this.filters.iterator();

      while(iterator.hasNext()) {
         Filter filter;

         filter = (Filter)iterator.next();
         object = filter.doFilter(object);
      }

      return object;
   }
}
```

7.8 Implementación del Configurador del Framework

El configurador del framework consiste de las siguientes clases:

- *ConfigXMLHelper*. Clase que obtiene las configuraciones definidas en el archivo XML.
- *XMLFactory*. Fábrica de datos XML, el cual transforma datos XML en objetos.
- *XMLIzable*. Interfaz que contiene las funciones que se deben implementar para transformar datos XML en objetos.

7.8.1 ConfigXMLHelper

```
/**
 * @(#)com.networkagent.framework.config.ConfigXMLHelper.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  17 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.config;

/**
* Clase que es utilizada por la fáblica de datos XML (XMLFactory)
* la cual obtiene las clases configuradas para la aplicación que
* extiende al framework. Las clases configuradas pertenecen a las
* secciones obligatorias y adicionales de cada una de las capas
* Agente, MIB y AVH.
*/

import java.util.Vector;
import java.lang.Class;
import java.lang.IllegalAccessException;
import java.lang.InstantiationException;
import java.lang.IllegalArgumentException;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Constructor;

import org.xml.sax.Attributes;

import com.networkagent.framework.util.FilterChain;
import com.networkagent.framework.util.Filter;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.ClassesLoader;
import com.networkagent.framework.agent.AdministratorDirVO;

public class ConfigXMLHelper implements XMLIzable
{
   private AgentResources  agentResources;
   private ClassesLoader   classesLoader;
   private FilterChain filterChain;
   private String          before;
   private Vector          data;

   public ConfigXMLHelper() {
      this.data           = new Vector();
      this.agentResources = AgentResources.getInstance();
      this.classesLoader  = ClassesLoader.getInstance();
   }

   public boolean setAttribute(String elementName, Attributes attrs) {
      String     auxiliarClass = "";
      Constructor   auxiliarConstructor;
      Object[]      parameters;
      String     className;

      try {
         if (elementName.equals(AgentResourcesKeys.MESSAGE_DECODER) == true) {
            auxiliarClass = attrs.getValue(0);
            this.classesLoader.setClass(AgentResourcesKeys.MESSAGE_DECODER, auxiliarClass);
         } else if (elementName.equals(AgentResourcesKeys.MESSAGE_RECEPTOR)) {
            auxiliarClass       = attrs.getValue(0);
            auxiliarConstructor = Class.forName(auxiliarClass).getConstructors()[0];
            parameters          = new Object[2];
            parameters[0]       = this.classesLoader.getObject(
                                  AgentResourcesKeys.MESSAGE_DECODER);
```

```
      parameters[1]       = new Integer(attrs.getValue(1));

      this.agentResources.setResource(AgentResourcesKeys.MESSAGE_RECEPTOR,
                                      auxiliarConstructor.newInstance(parameters));
   } else if (elementName.equals(AgentResourcesKeys.AGENT_FILTERCHAIN)) {
      this.filterChain = new FilterChain();
      this.agentResources.setResource(AgentResourcesKeys.AGENT_FILTERCHAIN, filterChain);
   } else if (elementName.equals(AgentResourcesKeys.FILTER)) {
      auxiliarClass = attrs.getValue(0);
      this.filterChain.addFilter((Filter) Class.forName(auxiliarClass).newInstance());
   } else if (elementName.equals(AgentResourcesKeys.AGENT_QUEUE)) {
      auxiliarClass       = attrs.getValue(0);
      auxiliarConstructor = Class.forName(auxiliarClass).getConstructors()[0];
      parameters    = new Object[2];
      parameters[0] = new Integer(attrs.getValue(1));
      parameters[1] = this.classesLoader.getObject(AgentResourcesKeys.AGENT_FILTERCHAIN);

      this.agentResources.setResource(AgentResourcesKeys.AGENT_QUEUE,
                                      auxiliarConstructor.newInstance(parameters));
   } else if (elementName.equals(AgentResourcesKeys.PROCESS_MANAGER)) {
      auxiliarClass       = attrs.getValue(0);
      auxiliarConstructor = Class.forName(auxiliarClass).getConstructors()[0];
      parameters          = new Object[1];
      parameters[0]       = new Integer(attrs.getValue(1));

      this.agentResources.setResource(AgentResourcesKeys.PROCESS_MANAGER,
                                      auxiliarConstructor.newInstance(parameters));
   } else if (elementName.equals(AgentResourcesKeys.SERVICE)) {
      auxiliarClass   = attrs.getValue(0);
      this.agentResources.setResource(auxiliarClass,
                                      Class.forName(attrs.getValue(1)).newInstance());
   } else if (elementName.equals(AgentResourcesKeys.MESSAGE_TRANSMISOR)) {
      auxiliarClass = attrs.getValue(0);
      this.classesLoader.setClass(AgentResourcesKeys.MESSAGE_TRANSMISOR, auxiliarClass);
   } else if (elementName.equals(AgentResourcesKeys.MESSAGE_CODER)) {
      auxiliarClass = attrs.getValue(0);
      this.classesLoader.setClass(AgentResourcesKeys.MESSAGE_CODER, auxiliarClass);
   } else if (elementName.equals(AgentResourcesKeys.ADMINISTRATORS)) {
      this.agentResources.setResource(AgentResourcesKeys.ADMINISTRATORS, new Vector());
   } else if (elementName.equals(AgentResourcesKeys.ADMINISTRATOR)) {
      Vector administrators = (Vector) this.agentResources.getResource(
                                       AgentResourcesKeys.ADMINISTRATORS);

      AdministratorDirVO administrator = new AdministratorDirVO(attrs.getValue(0),
                                         Integer.parseInt(attrs.getValue(1)));
      administrators.add(administrator);
   } else if (elementName.equals(AgentResourcesKeys.RESOURCE)) {
      auxiliarClass = attrs.getValue(1);
      this.agentResources.setResource(attrs.getValue(0),
                                      Class.forName(auxiliarClass).newInstance());
   } else if (elementName.equals(AgentResourcesKeys.ADD_CLASS)) {
      auxiliarClass = attrs.getValue(1);
      this.agentResources.setResource(attrs.getValue(0), Class.forName(auxiliarClass));
   } else if (elementName.equals(AgentResourcesKeys.MIB_CONTAINER)) {
      auxiliarClass = attrs.getValue(0);
      this.agentResources.setResource(AgentResourcesKeys.MIB_CONTAINER,
                                      Class.forName(auxiliarClass).newInstance());
   } else if (elementName.equals(AgentResourcesKeys.MIB_FILTERCHAIN)) {
      this.filterChain = new FilterChain();
      this.agentResources.setResource(AgentResourcesKeys.MIB_FILTERCHAIN,
                                      this.filterChain);
   } else if (elementName.equals(AgentResourcesKeys.MIB_QUEUE)) {
      auxiliarClass       = attrs.getValue(0);
      auxiliarConstructor = Class.forName(auxiliarClass).getConstructors()[0];
      parameters    = new Object[2];
      parameters[0] = new Integer(attrs.getValue(1));
      parameters[1] = this.classesLoader.getObject(AgentResourcesKeys.MIB_FILTERCHAIN);
      this.agentResources.setResource(AgentResourcesKeys.MIB_QUEUE,
                                      auxiliarConstructor.newInstance(parameters));
   } else if (elementName.equals(AgentResourcesKeys.AVH_MANAGER)) {
```

```
            this.agentResources.setResource(AgentResourcesKeys.SLEEP_TIME, attrs.getValue(0));


         } else if (elementName.equals(AgentResourcesKeys.AVH_CONTAINER)) {
            auxiliarClass = attrs.getValue(0);
            this.agentResources.setResource(AgentResourcesKeys.AVH_CONTAINER,
                                            Class.forName(auxiliarClass).newInstance());
         } else if (elementName.equals(AgentResourcesKeys.RESOURCES_ITERATOR)) {
            auxiliarClass = attrs.getValue(0);
           this.agentResources.setResource(AgentResourcesKeys.RESOURCES_ITERATOR,
                                            Class.forName(auxiliarClass).newInstance());
         } else if (elementName.equals(AgentResourcesKeys.REPORT_CREATOR)) {
            auxiliarClass  = attrs.getValue(0);
            this.agentResources.setResource(AgentResourcesKeys.REPORT_CREATOR,
                                            Class.forName(auxiliarClass).newInstance());
         }
      } catch (IllegalAccessException illexc) {
         System.out.println ("Error al intentar configurar la clase " + auxiliarClass +
                             ". Excepción: " + illexc.toString());
      } catch (InstantiationException instexc) {
         System.out.println ("Error al intentar configurar la clase " + auxiliarClass+
                             ". Excepción: " + instexc.toString());
      } catch (IllegalArgumentException illargexc) {
         System.out.println ("Error al intentar configurar la clase " + auxiliarClass +
                             ". Excepción: " + illargexc.toString());
      } catch (InvocationTargetException invexc) {
         System.out.println ("Error al intentar configurar la clase " + auxiliarClass +
                             ". Excepción: " + invexc.toString());
      } catch (ClassNotFoundException cnfexc) {
         System.out.println ("Error al intentar configurar la clase " + auxiliarClass +
                             ". Excepción: " + cnfexc.toString());
      }

      this.before = elementName;
      return true;
   }

   public boolean setAttribute(String value) {
      return true;
   }

   public void signal(String elementName) {

   }

   public Vector getData() {
      return this.data;
   }
}
```

7.8.2 XMLFactory

```
/**
 * @(#)com.networkagent.framework.config.XMLFactory.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  16 de Noviembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.config;

/**
* Clase que implementa el mecanismo de interpretación de flujos y
* archivos XML, el cual devuelve como resultado un Vector que
* representa los datos obtenidos del XML.
*/

import com.networkagent.framework.config.XMLIzable;

import java.io.IOException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Vector;
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXNotRecognizedException;

public class XMLFactory extends DefaultHandler
{
   private     Vector     valueObjects;
   private     String     xmlHelperClassName;
   private     XMLIzable      xmlizable;

   public Vector createXMLData(String xmlFile, Class xmlValueObjectHelperClass)
                throws SAXNotRecognizedException, SAXException, IOException {
      SAXParser parser = new SAXParser();
      parser.setFeature("http://xml.org/sax/features/validation", true);
      parser.setContentHandler(this);
      parser.setErrorHandler(this);
      parser.setDTDHandler(this);
      createXMLizable(xmlValueObjectHelperClass);
      parser.parse(xmlFile);

      return xmlizable.getData();
   }

   public Vector createXMLData(InputStream stream, Class xmlValueObjectHelperClass)
                throws SAXNotRecognizedException, SAXException, IOException {
      SAXParser parser = new SAXParser();
      parser.setFeature("http://xml.org/sax/features/validation", true);
      parser.setContentHandler(this);
      parser.setErrorHandler(this);
      parser.setDTDHandler(this);
      createXMLizable(xmlValueObjectHelperClass);
      parser.parse(new InputSource(stream));

      return xmlizable.getData();
   }

   public void startElement(String url, String local, String qName, Attributes attributes) {
      xmlizable.setAttribute(local, attributes);
   }
```

```
   public void endElement(String url, String local, String qName) {
      xmlizable.signal(local);
   }

   public void characters(char[] chars, int start, int length) {
      String string = new String(chars, start, length).trim();

      if (string.length() > 0) {
         xmlizable.setAttribute(string);
      }
   }

   public void warning(SAXParseException ex) {
      System.err.println("[Warning] " + ": " + ex.getMessage());
   }

   public void error(SAXParseException ex) {
      System.err.println("[Error] " + ": " + ex.getMessage());
   }

   public void fatalError(SAXParseException ex) throws SAXException {
      System.err.println("[Fatal Error] " + ": " + ex.getMessage());
      throw ex;
   }

   private void createXMLizable(Class xmlValueObjectHelperClass) {
      try {
         xmlizable = (XMLIzable)xmlValueObjectHelperClass.newInstance();
      } catch(InstantiationException iexc) {
         System.out.println("No se pudo instanciar de la clase: " +
         xmlValueObjectHelperClass.getName() + " excepción: " + iexc.getMessage());
      } catch(IllegalAccessException iaexc) {
         System.out.println("No se pudo instanciar de la clase: " +
         xmlValueObjectHelperClass.getName() + " excepción: " + iaexc.getMessage());
      }
   }
}
```

7.8.3 XMLIzable

```
/**
 * @(#)com.networkagent.framework.config.XMLIzable.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  16 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.framework.config;

/**
* Interfaz utilizada en el mecanismo de conversión de flujo XML
* a datos legibles por la aplicación.
*/

import java.util.Vector;
import org.xml.sax.Attributes;

public interface XMLIzable
{
   public abstract boolean setAttribute(String elementName, Attributes attrs);
   public abstract boolean setAttribute(String value);
   public abstract void signal(String elementName);
   public abstract Vector getData();
}
```

Capítulo

8

Aplicación del Framework: Agente Proxy SNMP para el Ruteador Cisco 2501

8.1 Introducción

En el presente capítulo se desarrolla una aplicación que utiliza el framework de dominio, la cual representa a un Agente Proxy para el Ruteador Cisco 2501. Cabe destacar que el Agente utiliza a SNMP como protocolo de administración de red.

El capítulo comienza describiendo la fase de desarrollo de la aplicación, en la cual se describen los pasos necesarios para generar cualquier aplicación Agente que extienda del framework.

Después se presentan cada una de las actividades involucradas en la fase de desarrollo. Esta fase consiste de las siguientes actividades:

- *Propuesta de Desarrollo*. Describe los objetivos, funciones y servicios con los que cuenta la aplicación.
- *Análisis de Requerimientos*. Donde se define para cada capa del Agente, las características con las que deben contar los objetos del dominio.
- *Diseño de la Aplicación*, En donde se puede observar la extensibilidad del framework a través de la aplicación.
- *Implementación*. La cual presenta la codificación de las clases Java de la aplicación.

Por último se dan las conclusiones del presente capítulo.

8.2 Fase de Desarrollo de la Aplicación

La fase de desarrollo de la aplicación contiene varias actividades a realizar, las cuales tienen como fin el validar al framework de dominio. Todas las actividades de esta fase se encuentran plasmadas en la Figura 8.1.

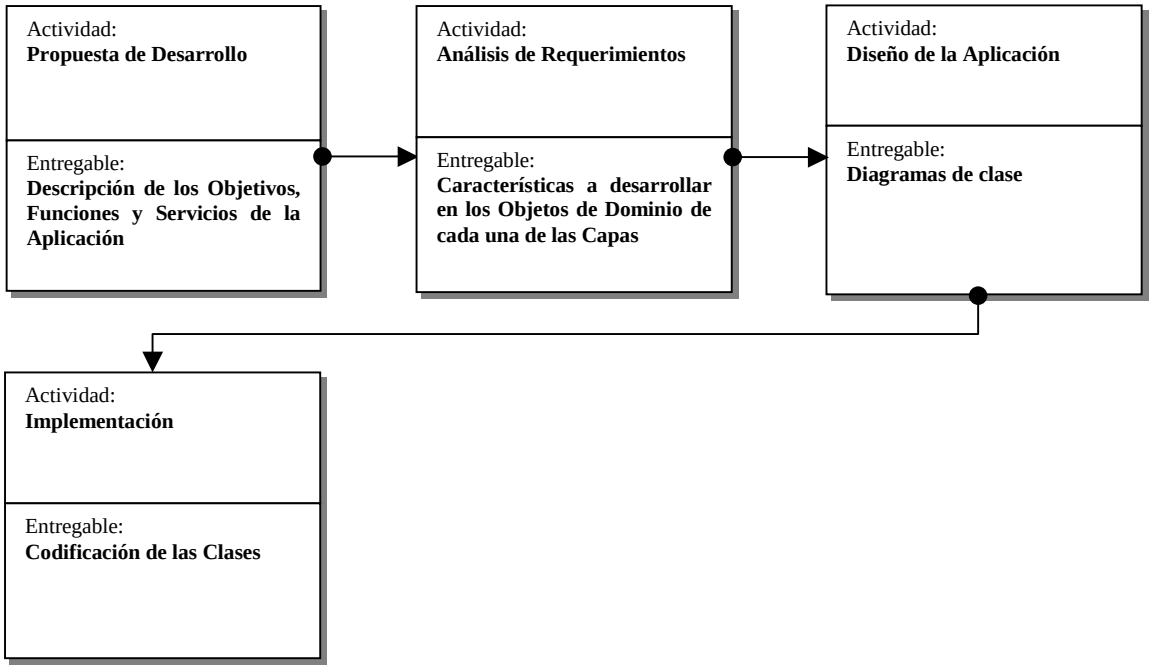


Figura 8.1 Fase de Desarrollo de la Aplicación.

Las actividades a desarrollar son las siguientes:

- **Propuesta de Desarrollo**
  Consiste en describir a grandes rasgos la funcionalidad, las características y servicios con las que contará la aplicación.
  Entregable
  El entregable en esta actividad es una descripción de objetivos, funciones y servicios proporcionados por la aplicación.

- **Análisis de Requerimientos**
  Los requerimientos de la aplicación son plasmados y agrupados en la definición de las características de los objetos de dominio que corresponden a cada una de las capas *Agente*, *MIB*, y *AVH*.
  Entregable
  Características de los Objetos de Dominio que se deben implementar en las clases de la aplicación.

- **Diseño de la Aplicación**
  Las características de los Objetos de Dominio son mapeadas clases, las cuales representan la extensibilidad del framework.
  Entregable
  Diagramas de Clase de la aplicación.

- **Implementación**
  Las clases presentadas en el diseño son codificadas en el lenguaje Java.
  Entregable
  Codificación de las clases de la aplicación a través del lenguaje Java.

8.3 Propuesta de Desarrollo

Para la validación del Framework de Dominio se propone desarrollar una aplicación que implemente un Agente Proxy SNMP para el Ruteador Cisco 2501. La aplicación es un Agente Proxy ya que para obtener el estado de elementos del ruteador (tales como conexiones, interfaces, enlaces, etc.) no se realiza a través de instrucciónes de bajo nivel, sino a través de una sesión telnet que permite el envío de comandos del Sistema Operativo IOS al Ruteador. Como puede intuirse, la aplicación Agente no se encuentra en el Ruteador, sino en una PC conectada a la red configurada en el mismo.

Se propone además desarrollar un simulador de Administrador de Red, el cual envíe en intervalos de tiempo variables comandos del protocolo SNMP, para que sean interpretados y ejecutados en su representación.

La aplicación será limitada en cuanto a la interfaz gráfica, ya que el resultado del monitoreo de los Recursos del Ruteador, y de la ejecución de los comandos SNMP, deberá desplegarse en modo texto a través de la salida estandar del Sistema Operativo MS-DOS.

Los Objetos Administrados de la aplicación corresponderán al grupo de administración que permite tener información respecto a las interfaces del ruteador Ethernet 0, Serial 0 y Serial 1.

Se propone por último que los Recursos del Ruteador que sean monitoreados correspondan también al estado de las interfaces Ethernet 0, Serial 0 y Serial 1 del ruteador.

La Figura 8.2 muestra un esquema en donde participan los elementos de la aplicación propuesta y sus conexiones.

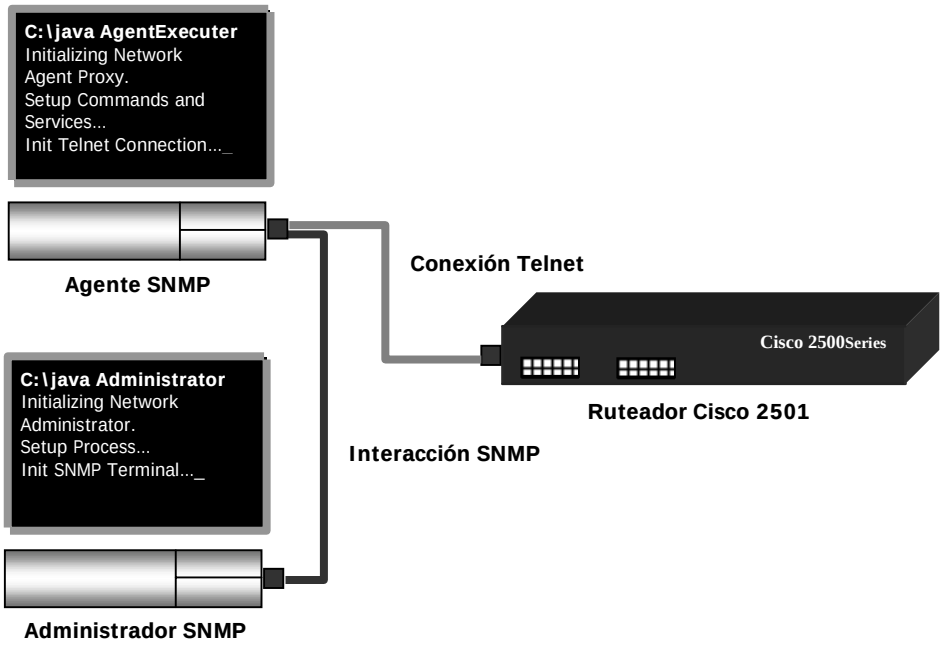


Figura 8.2 Aplicación Proxy SNMP para el Ruteador Cisco 2501.

8.4 Análisis de Requerimientos

A través del Análisis de Requerimientos de la aplicación es posible determinar la funcionalidad esperada por la aplicación, así como las características generales con las que deben contar los Componentes del Dominio también llamadas capas.

A continuación para cada una de las capas se determinan los requerimientos de la aplicación.

8.4.1 Capa Agente

Los requerimientos de la aplicación para la capa de Agente son:

- *Protocolo SNMP*. Se debe utilizar SNMP como protocolo de administración de red, para el cual se aceptarán los siguientes comandos:
  - *GET-REQUEST*. Comando utilizado para la obtención de información de Objetos Administrados.
  - *TRAP*. Comando que permite reportar al Administrados cambios de estado en los recursos.
- *Recepción y Envío de Mensajes a través de UDP*. Debido a que el protocolo SNMP no es orientado a conexión, se utiliza entonces el protocolo UDP para la recepción y envío de mensajes.

- *Decodificador de Mensajes BER*. Tanto el codificador como el decodificador de mensajes utilizará el esquema de codificación BER para transformar el flujo de datos en Mensajes y viceversa.
- *Cola de Mensajes FIFO*. La cola de mensajes del Agente corresponderá al esquema FIFO (first in-first out), es decir los mensajes se despacharán de acuerdo a su llegada.
- *Servicio de Filtrado de Eventos*. El servicio de filtrado de eventos vigilará, que los reportes de eventos cumplan con ciertos requerimientos, como el permtir enviar o no reportes de ciertos recursos en particular.
- *Administrador de Procesos con Pool*. El Administrador de procesos utilizará un pool de procesos para despachar los servicios solicitados.
- *Implementación de Servicios GET-REQUEST y TRAP*. Debido a que el Agente debe soportar la ejecución de los comandos GET-REQUEST y TRAP, entonces se deben implementar sus correspondientes servicios.

8.4.2 Capa MIB

Los requerimientos de la aplicación para la capa de Agente son:

- *Contenedor de Base de Datos para Objetos Administrados*. El contenedor de Objetos Administrados tiene que mapear las solicitudes y actualizaciones de información de Objetos Administrados a sus correspondientes tablas y comandos de SQL. La Base de Datos es implementada a través de *Microsoft Access*.
- *Cola de Mensajes FIFO*. Al igual que la cola de mensajes del Agente, la cola de mensajes de la MIB corresponderá al esquema FIFO.
- *Cadena de Filtros Vacía*. Debido a que la aplicación corresponde a un Agente SNMP, no se requiere implementar ningún filtro que discrimine los reportes de cambio de estado en los recursos.
- *Objetos Administrados*. Los Objetos Administrados a implementar pertenecen al siguiente grupo de administración:
    - *lifTable*. Grupo Local de Interfaces el cual permite mantener información referente a interfaces del Ruteador Cisco 2501 (Este grupo se encuentra definido en el ASN CISCO-INTERFACES-MIB, el cual se puede obtener del sitio web de Cisco Systems [CISCO, 1994]). Cabe destacar que no toda la tabla será implementada, sino solamente algunos elementos que sean relevantes.

A continuación se presenta la definición del grupo de administración *lifTable* a través de la Notación de Sintaxis Abstracta (ASN de sus siglas en Inglés).

Grupo lifTable

```
CISCO-INTERFACES-MIB DEFINITIONS ::= BEGIN

IMPORTS      Counter                   FROM RFC1155-SMI
             OBJECT-TYPE               FROM RFC-1212
             DisplayString, ifIndex    FROM RFC1213-MIB
             local                     FROM CISCO-SMI;

linterfaces OBJECT IDENTIFIER ::= { local 2 }

-- Local Interface Group. This group is present in all products.
-- Local Interface Table
-- This group provides additional objects to the table defined by RFC1156.

lifTable     OBJECT-TYPE
             SYNTAX       SEQUENCE OF LifEntry
             ACCESS       not-accessible
             STATUS       mandatory
             DESCRIPTION  "A list of interface entries."
                       ::= { linterfaces 1 }

lifEntry     OBJECT-TYPE
             SYNTAX       LifEntry
             ACCESS       not-accessible
             STATUS       mandatory
             DESCRIPTION  "A collection of additional objects in the cisco interface."
                          INDEX { ifIndex }
                          ::= { lifTable 1 }

lifEntry ::= SEQUENCE {    locIfHardType          DisplayString,
                        locIfLineProt          INTEGER,
                           locIfLineInt              INTEGER,
                           locIfInRunts              INTEGER,
                           locIfInGiants          INTEGER,
                           locIfInCRC                INTEGER,
                           locIfInFrame              INTEGER,
                           locIfInOverrun            INTEGER,
                           locIfInIgnored            INTEGER,
                          locIfInAbort               INTEGER,
                        locIfResets               INTEGER,
                        locIfRestarts          INTEGER,
                        locIfKeep                 INTEGER,
                        locIfReason               DisplayString,
                        locIfCarTrans          INTEGER,
                           locIfCollisions           INTEGER,
                          locIfDescr             DisplayString,
                           locIfipInPkts          Counter,
                          locIfipOutPkts             Counter,
                          locIfipInOctects           Counter,
                          locIfipOutOctects          Counter   }
-- The following section describes the components of the table.

locIfHardType       OBJECT-TYPE
SYNTAX              DisplayString
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Returns the type of interface."
                    ::= { lifEntry 1 }

locIfLineProt       OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Boolean whether interface line protocol is up or not."
                    ::= { lifEntry 2 }
```

```
locIfLineInt     OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Boolean whether the interface is up or not."
                    ::= { lifEntry 3 }

locIfInRunts     OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Number of packets input which were smaller
                    then the allowable physical media permitted."
                   ::= { lifEntry 10 }

locIfInGiants       OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Number of input packets which were larger
                    then the physical media permitted."
                    ::= { lifEntry 11 }

locIfInCRC          OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Number of input packets which had cyclic
                    redundancy checksum errors."
                    ::= { lifEntry 12 }

locIfInFrame     OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Number of input packet which were
                    misaligned."
                   ::= { lifEntry 13 }

locIfInOverrun      OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Count of input which arrived too quickly for
                    the to hardware receive."
                   ::= { lifEntry 14 }
locIfInIgnored      OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "Number of input packets which were simply
                    ignored by this interface."
                   ::= { lifEntry 15 }

locIfInAbort     OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Number of input packets which were aborted."
                    ::= { lifEntry 16 }

locIfResets         OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Number of times the interface internally reset."
                   ::= { lifEntry 17 }
```

```
locIfKeep           OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "Boolean whether keepalives are enabled on this interface."
                    ::= { lifEntry 19 }

locIfCollisions     OBJECT-TYPE
SYNTAX              INTEGER
ACCESS              read-only
STATUS              mandatory
DESCRIPTION         "The number of output collisions detected on this interface."
                    ::= { lifEntry 25 }

locIfDescr          OBJECT-TYPE
SYNTAX              DisplayString
ACCESS              read-write
STATUS              mandatory
DESCRIPTION         "User configurable interface description."
                    ::= { lifEntry 28 }

locIfipInPkts       OBJECT-TYPE
SYNTAX              Counter
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "ip protocol input packet count"
                   ::= { lifEntry 42 }

locIfipOutPkts      OBJECT-TYPE
SYNTAX              Counter
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "ip protocol output packet count"
                   ::= { lifEntry 43 }

locIfipInOctets     OBJECT-TYPE
SYNTAX              Counter
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "ip protocol input octet count"
                   ::= { lifEntry 44 }

locIfipOutOctets    OBJECT-TYPE
SYNTAX              Counter
ACCESS              read-only
STATUS              mandatory
DESCRIPTION        "ip protocol output octet count"
                   ::= { lifEntry 45 }
-- End of table
```

8.4.3 Capa AVH

Los requerimientos para la capa de Administración Virtual de Hardware son:

- *Contenedor de Recursos*. El Contenedor de Recursos del Elemento de Red crea e inicializa los Recursos.
- *Monitoreo Aleatorio de Recursos*. El monitoreo de recursos se realiza de manera aleatoria, es decir, el iterador debe implementar un mecanismo aleatorio para la obtención de los Recursos.
- *Reporteador con soporte para generación de Traps*. El reporteador de eventos debe contar con el soporte necesario para generar los siguientes traps:
- *Recursos*. Los recursos que deben ser monitoreados son los siguientes:
  - *Interfaz Ethernet 0*. Interface que permite interconectar elementos de red como switches, hubs, etc.
  - *Interfaz Serial 0*. Interface que permite interconectar a ruteadores.
  - *Interfaz Serial 1*. Interface que permite interconectar a ruteadores.

Una vez determinados los requerimientos del Agente en cada una de sus capas, se presenta el diseño de las clases de la aplicación.

8.5 Diseño de la Aplicación

8.5.1 Capa Agente

La funcionalidad de la aplicación en su capa Agente se encuentra definida en las siguientes secciones:

1. *Recepción de Mensajes*. En esta sección se define la forma como se reciben los flujos de datos enviados por el Administrador y se transforman en mensajes SNMP.
2. *Mensajes SNMP Soportados por la Aplicación*. En esta sección se definen los mensajes soportados por la aplicación, los cuales son GET-REQUEST y TRAP.
3. *Mecanismo de Contención de Mensajes*. En el se define el mecanismo FIFO de contención de mensajes utilizado por la aplicación.
4. *Ejecución de Servicios*. Los servicios soportados por la aplicación son definidos en esta sección, los cuales son el servicio GET-REQUEST y TRAP de SNMP.
5. *Transmisión de Mensajes*. Por último, en esta sección se define la forma en que se codifican y envían los mensajes SNMP del Agente al Administrador.

8.5.1.1 Recepción de Mensajes

La Figura 8.3 muestra el mecanismo de recepción de mensajes de la aplicación, la cual es representada por la funcionalidad definida en las clases *UDPMessageReceptor* y *SNMPMessageDecoder*.

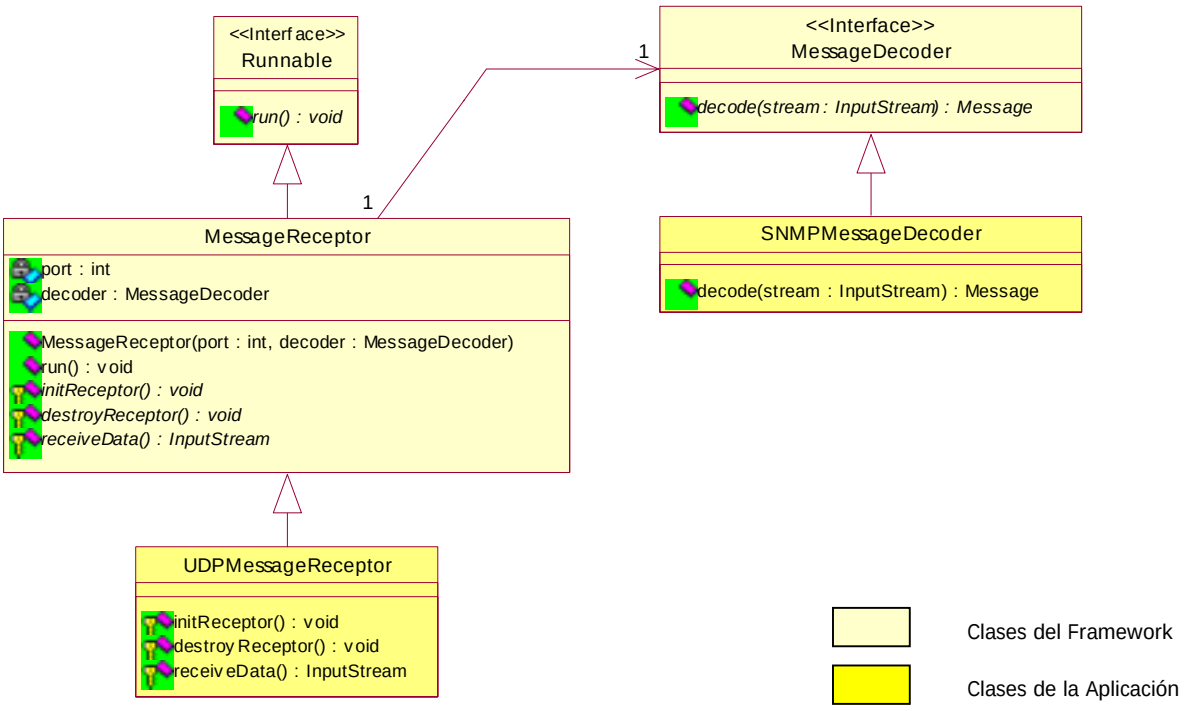


Figura 8.3 Mecanismo de Recepción de Mensajes de la Aplicación Agente.

Las clases que son implementadas en la aplicación son:

- ***UDPMessageReceptor***. Clase que extiende de *MessageReceptor* y que permite recibir datos a través del protocolo UDP (User Datagram Protocol). Se implementó la recepción de datos a través de UDP debido a que SNMP es un protocolo no orientado a conexión. Los métodos que se concretizan de la clase *MessageReceptor* son:
  - ***initReceptor(…)***. Inicia el mecanismo de obtención de mensajes, el cual es un Servidor de Datagramas UDP, utilizando como puerto de escucha 161 (Puerto de escucha por definición del protocolo SNMP).
  - ***destroyReceptor(…)***. Finaliza el mecanismo de obtención de mensajes, en este caso libera los recursos utilizados por el Servidos de Datagramas UDP.
  - ***receiveData(…)***. Permite la recepción de un flujo de datos. Este método utiliza al Servidor de Datagramas UDP para recibir un flujo de datos, para después utilizar el mecanismo de decodificación de mensajes determinado por la aplicación (En este caso implementado por la clase *SNMPMessageDecoder*).

- ***SNMPMessageDecoder***. Clase que extiende de *MessageDecoder* y que permite transformar un flujo de datos en un mensaje SNMP configurado en la aplicación Agente. Cabe destacar que en este caso el único tipo de mensaje que se configura en la

aplicación es el mensaje GET-REQUEST. Los métodos que se concretizan de la clase *MessageReceptor* son:

- ***decode(…)***. Transforma un flujo de datos en un mensaje SNMP configurado en la aplicación.

8.5.1.2 Mensajes SNMP Soportados por la Aplicación

La Figura 8.4 muestra los mensajes SNMP configurados en la aplicación, los cuales se encuentran definidos por las clases *SNMPTrapMessage* y *SNMPGetMessage*.

Figura 8.4 Mensajes SNMP Soportadas por la Aplicación.

Las clases implementadas en la aplicación son:

- ***SNMPTrapMessage***. Clase que extiende de *Message* y que encapsula los datos referentes a un mensaje TRAP de SNMP. Ningún método es concretizado de la clase *Message*, mas sin embargo se añaden los siguientes métodos:
  - ***getEnterprise(…)***. Obtiene el identificador del Objeto que generó la notificación TRAP.
  - ***getAgentAddress(…)***. Obtiene la dirección IP del Agente.
  - ***getGenericTrap(…)***. Obtiene el tipo de mensaje TRAP enviado al Administrador, el cual es especificado por SNMP.
  - ***getSpecificTrap(…)***. Obtiene el tipo de mensaje TRAP enviado al Administrador, el cual es especificado de manera propietaria.
  - ***getTimeStamp(…)***. Obtiene el tiempo de generación del mensaje TRAP.

- ***getVarBindList(…)***. Obtiene la lista de valores de Objetos Administrados relacionados al mensaje TRAP.
- ***setEnterprise(…)***. Asigna el identificador del Objeto que generó la notificación TRAP.
- ***getAgentAddress(…)***. Asigna la dirección IP del Agente.
- ***getGenericTrap(…)***. Asigna el tipo de mensaje TRAP enviado al Administrador, el cual es especificado por SNMP.
- ***getSpecificTrap(…)***. Asigna el tipo de mensaje TRAP enviado al Administrador, el cual es especificado de manera propietaria.
- ***getTimeStamp(…)***. Asigna el tiempo de generación del mensaje TRAP.
- ***getVarBindList(…)***. Asigna la lista de valores de Objetos Administrados relacionados al mensaje TRAP.

- ***SNMPGetMessage***. Clase que extiende de *Message* y que encapsula los datos referentes a un mensaje GET-REQUEST de SNMP. Ningún método es concretizado de la clase *Message*, mas sin embargo se añaden los siguientes métodos:
  - ***getVersion(…)***. Obtiene la versión del protocolo SNMP.
  - ***getCommunityName(…)***. Obtiene el nombre de la comunidad SNMP a la que pertenece el Administrador y el Agente.
  - ***getStatusError(…)***. Obtiene el estatus de error del mensaje.
  - ***getErrorIndex(…)***. Obtiene el tipo de error presente en el mensaje.
  - ***getVarBindList(…)***. Obtiene la lista de valores de Objetos Administrados relacionados al mensaje.
  - ***setVersion(…)***. Asigna la versión del protocolo SNMP.
  - ***setCommunityName(…)***. Asigna el nombre de la comunidad SNMP a la que pertenece el Administrador y el Agente.
  - ***setStatusError(…)***. Asigna el estatus de error del mensaje.
  - ***setErrorIndex(…)***. Asigna el tipo de error presente en el mensaje.
  - ***setVarBindList(…)***. Asigna la lista de valores de Objetos Administrados relacionados al mensaje.

- ***VariablesBind***. Clase que representa Objetos Administrados contenidos en los mensajes SNMP. Los métodos de la clase *VariablesBind* son:
  - ***getInstance(…)***. Obtiene el identificador del Objeto Administrado.
  - ***getValue(…)***. Obtiene el valor del Objeto Administrado.
  - ***setInstance(…)***. Asigna el identificador del Objeto Administrado.
  - ***setValue(…)***. Asigna el valor del Objeto Administrado.

8.5.1.3 Mecanismo de Contención de Mensajes

La Figura 8.5 muestra el mecanismo de conteción de mensajes en la capa Agente, el cual se encuentra definido por las clases *FifoMessageQueue*, *CommunityFilter* y *EventsFilter*.

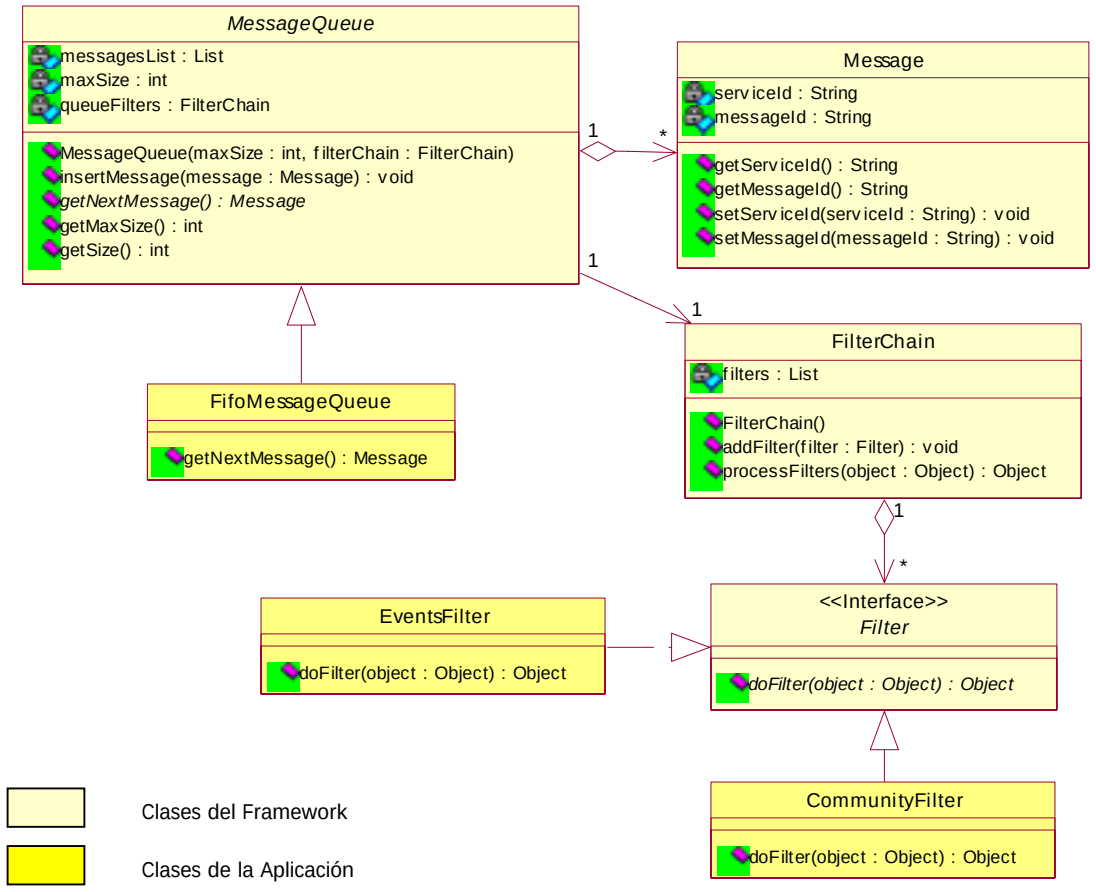


Figura 8.5 Mecanismo de Contención de Mensajes Implementado en la Aplicación.

Las clases implementadas en la aplicación son:

- ***FifoMessageQueue***. Clase que extiende de *MessageQueue*, la cual es un contenedor de mensajes de tipo FIFO (First In-Fist Out), es decir, cuando la instancia de la clase *Agent* la utilice para obtener el siguiente mensaje a analizar, se le retornará el primer mensaje que se encuentre en la lista de espera. Los métodos que se concretizan de la clase *MessageQueue* son:
  - ***getNextMessage(…)***. Obtiene el siguiente mensaje a analizar por la entidad Agente. En este caso, se implementa el mecanismo FIFO para obtener el mensaje de la lista de espera.

- ***CommunityFilter***. Clase que extiende de *Filter*, la cual verifica que los mensajes enviados al Agente pertenezcan a una misma comunidad SNMP. Los métodos que se concretizan de la interfaz *Filter* son:
  - ***doFilter(…)***. Discrimina un mensaje SNMP verificando la comunidad a la que pertenece el mismo.

- ***EventsFilter***. Clase que extiende de *Filter*, la cual discrimina los mensajes de reporte de eventos enviados por la AVH. En este caso solo permite enviar cambios de estado en las interfaces Ethernet 0, Serial 0 y Serial 1 del ruteador, no así de la lista de direcciones ARP. Los métodos que se concretizan de la interfaz *Filter* son:
  - ***doFilter(…)***. Discrimina un mensaje TRAP de reporte de cambio de estado, permitiendole al Agente solo procesar aquellos que pertenezcan a un cambio de estado en las interfaces Ethernet 0, Serial 0 y Serial 1 del ruteador.

8.5.1.4 Ejecución de Servicios

La Figura 8.6 muestra el mecanismo de ejecución de servicio así como los servicios definidos en el Agente, los cuales se encuentran definidos por las clases *PoolProcessManager*, *GetService* y *TrapService*.

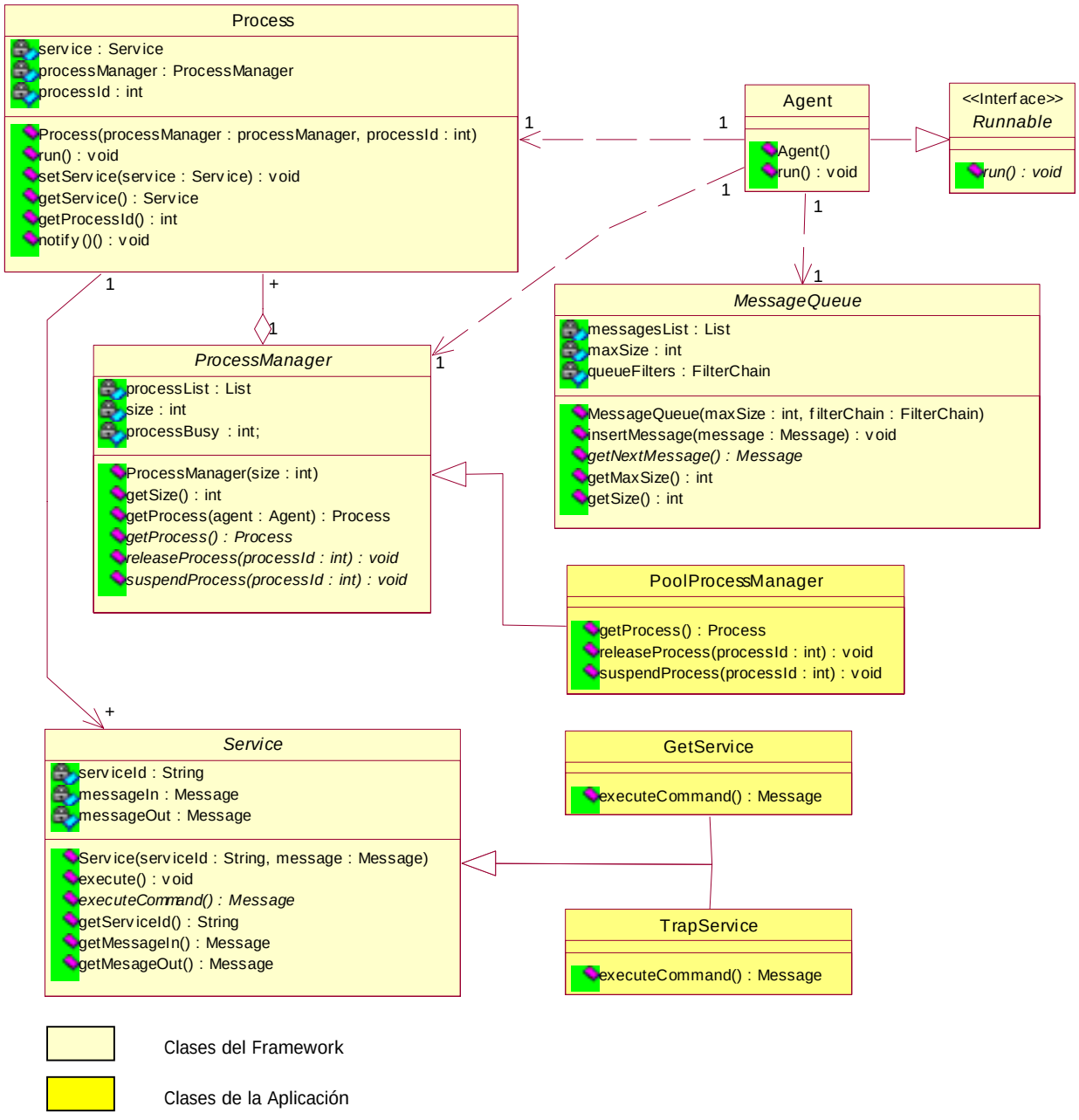


Figura 8.6 Ejecución y Definición de Servicios de la Aplicación.

Las clases implementadas en la aplicación son:

- ***PoolProcessManager***. Clase que extiende de *ProcessManager*, la cual implementa el pool de procesos que serán utilizados para la ejecución de servicios SNMP. Los métodos que se concretizan de la clase *ProcessManager* son:
  - ***getProcess(…)***. Obtiene un proceso disponible del pool.
  - ***releaseProcess(…)***. Libera y retorna el proceso al pool.
  - ***suspendProcess(…)***. Suspende la ejecución de un proceso y lo retorna al pool.

- ***GetService***. Clase que extiende de *Service*, la cual implementa el servicio GET-REQUEST del protocolo SNMP. Los métodos que se concretizan de la clase *Service* son:
  - ***executeCommand(…)***. Ejecuta la lógica de negocio necesaria para implementar el servicio GET-REQUEST en el Agente.

- ***TrapService***. Clase que extiende de *Service*, la cual implementa el TRAP del protocolo SNMP. Los métodos que se concretizan de la clase *Service* son:
  - ***executeCommand(…)***. Ejecuta la lógica de negocio necesaria para implementar el TRAP en el Agente.

8.5.1.5 Transmisión de Mensajes

La Figura 8.7 muestra el mecanismo de transmisión de mensajes utilizado por el Agente, el cual se encuentra definido por las clases *UDPMessageTransmisor* y *SNMPMessageCoder*.

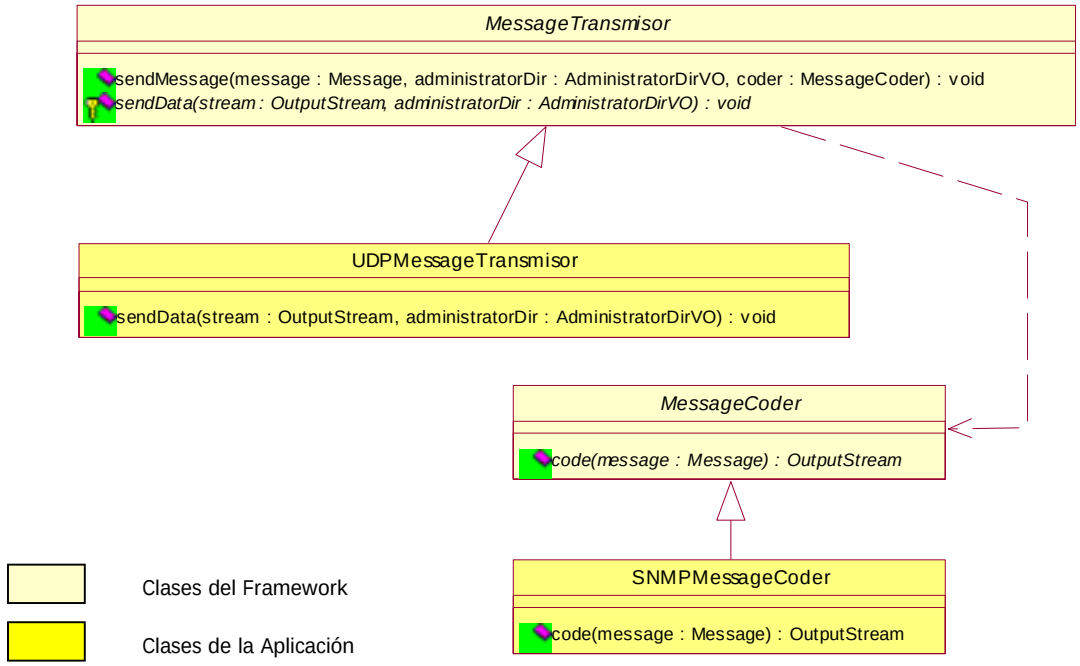


Figura 8.7 Transmisión y Codificación de Mensajes SNMP en la Aplicación.

Las clases implementadas en la aplicación son:

- ***UDPMessageTransmisor***. Clase que extiende de *MessageTransmisor* y que permite enviar datos al Administrador a través del protocolo UDP (User Datagram Protocol). Se implementó la transmisión de datos a través de UDP debido a que SNMP es un protocolo no orientado a conexión. Los métodos que se concretizan de la clase *MessageMessageTransmisor* son:
  - ***sendData(…)***. Permite la transmisión de un flujo de datos. Este método utiliza el mecanismo de Datagramas UDP para enviar un flujo de datos, para después utilizar el mecanismo de codificación de mensajes determinado por la aplicación (En este caso implementado por la clase *SNMPMessageCoder*).

- ***SNMPMessageCoder***. Clase que extiende de *MessageCoder* y que permite transformar un mensaje SNMP en flujo de datos. Los mensajes que pueden ser codificados son GET-RESPONSE y TRAP. Los métodos que se concretizan de la clase *MessageReceptor* son:
  - ***code(…)***. Transforma un mensaje SNMP en un flujo de datos.

8.5.2 Capa MIB

La funcionalidad de la aplicación en su capa MIB se encuentra definida en las siguientes secciones:

1. *Contención de Objetos Administrados y Despacho de Reportes de Cambio de Estado*. En esta sección se define el mecanismo utilizado para contener los reportes de cambio de estado enviados por el Administrador Virtual de Hardware, así como el contenedor utilizado para almacenar los Objetos Administrados de la aplicación, el cual es una Base de Datos.
2. *Objetos Administrados del Grupo lifTable*. En esta sección se definen las clases de Objetos Administrados correspondientes al grupo de interfaces locales (*lifTable*), el cual corresponde a las interfaces seriales y ethernet del ruteador.

8.5.2.1 Contención de Objetos Administrados y Despacho de Reportes de Cambio de Estado

La Figura 8.8 muestra el contenedor de Objetos Administrados utilizado en la aplicación, así como la cola de reportes de cambio de estado enviados por el AVH. Esta funcionalidad se encuentra definida en las clases *MIBDBContainer* y *FifoMessageQueue* respectivamente.

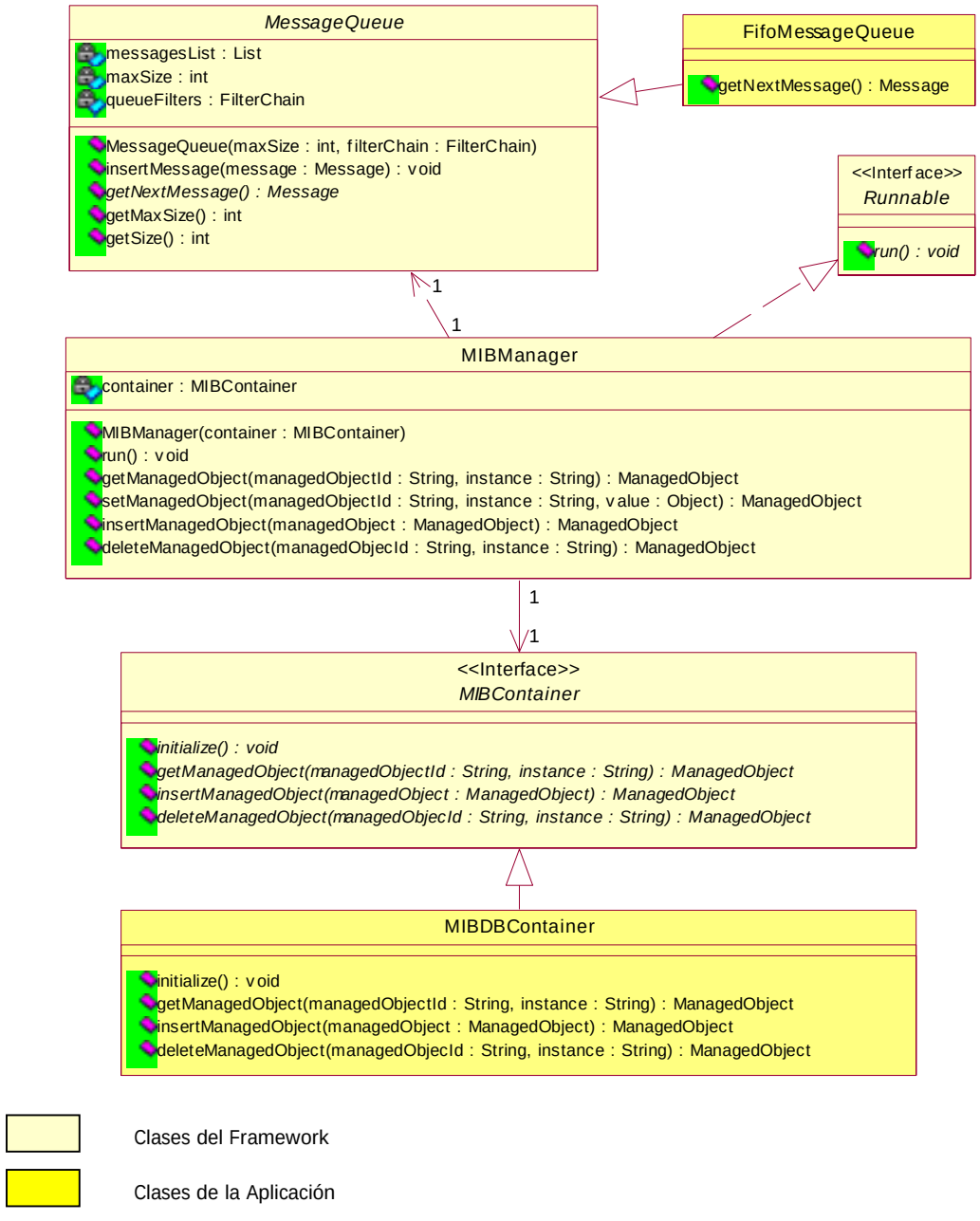


Figura 8.8 Contenedor de Objetos Administrados y Despacho de Reportes de Cambio de Estado.

Las clases implementadas en la aplicación son:

- ***MIBDBContainer***. Clase que extiende de *MIBContainer*, la cual implementa un mecanismo de contención de Base de Datos para los Objetos Administrados. Los métodos que se concretizan de la clase *MIBContainer* son:
  - ***initialize(…)***. Inicializa el contenedor de Objetos Administrados, creando conexiones a la Base de Datos además de verificar que los valores que se encuentran en la misma corresponden al estado actual de los recursos del ruteador, en este caso, la interfaces Ethernet 0, Serial 0 y Serial 1, además de las tablas ARP de translación.
  - ***getManagedObject(…)***. Obtiene el Objeto Administrado solicitado a través de la Base de Datos.
  - ***insertManagedObject(…)***. Añade un Objeto Administrado a la Base de Datos.
  - ***deleteManagedObject(…)***. Elimina un Objeto Administrado de la Base de Datos.

- ***FifoMessageQueue***. Clase que extiende de *MessageQueue*, la cual es un contenedor de mensajes de tipo FIFO (First In-Fist Out). Los métodos que se concretizan de la clase *MessageQueue* son:
  - ***getNextMessage(…)***. Obtiene el siguiente mensaje a analizar por la entidad Agente. En este caso, se implementa el mecanismo FIFO para obtener el mensaje de la lista de espera.

8.5.2.2 Objetos Administrados del Grupo lifTable

La Figura 8.9 muestra las clases de Objetos Administrados utilizados para implementar el grupo lifTable de SNMP. Estas clases son *LifTable* y *LifEntry*.

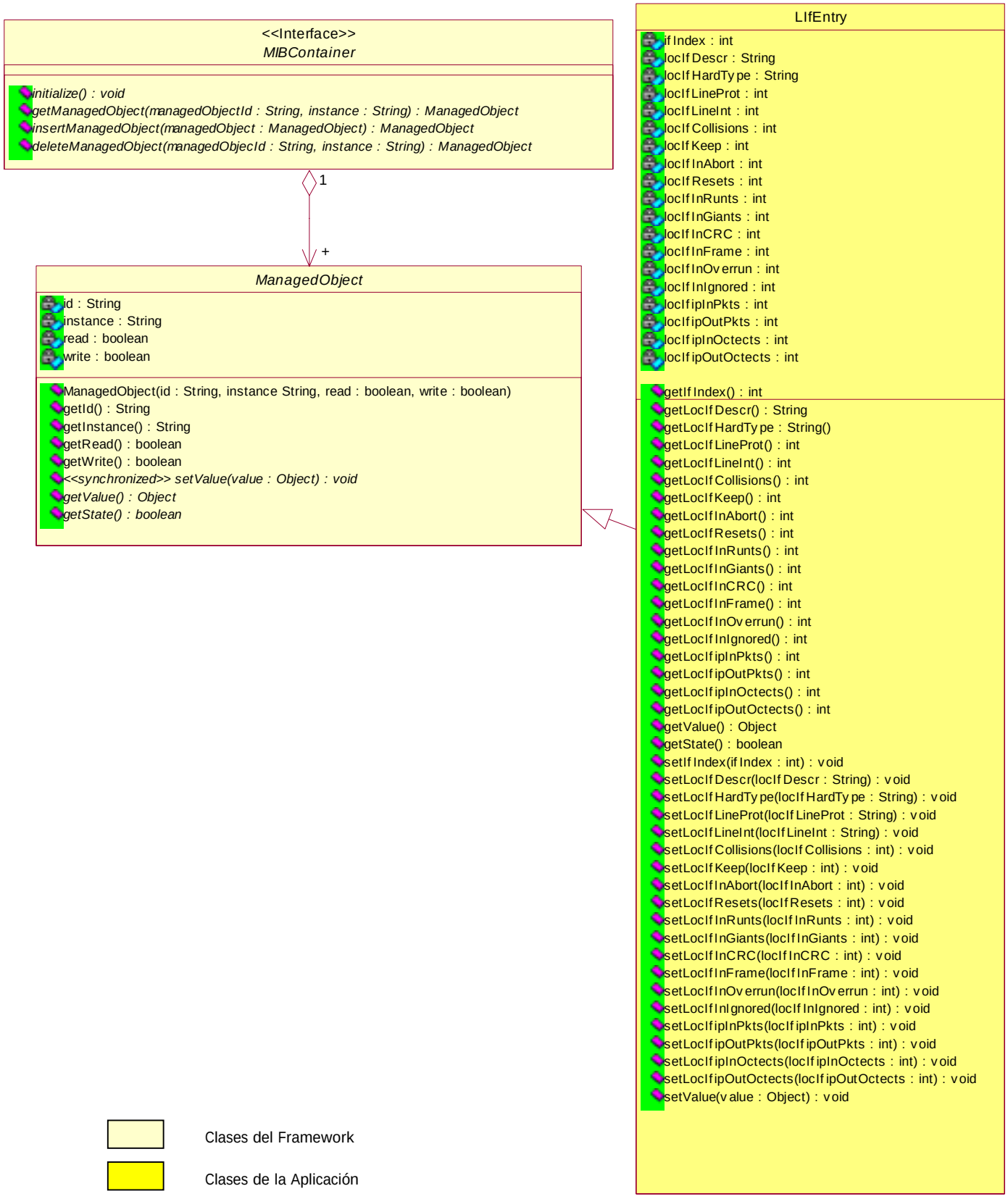


Figura 8.9 Objetos Administrados del Grupo lifTable.

Las clases implementadas en la aplicación son:

- ***LIfTable***. Clase que extiende de *ManagedObject*, la cual implementa a la tabla *ifTable* de SNMP. Los métodos que se concretizan de la clase *ManagedObject* son:
  - ***setValue(…)***. En este caso *setValue(…)* es vacía debido a que no se puede actualizar una tabla SNMP.
  - ***getValue(…)***. Obtiene la lista entera de elementos de la tabla, en este caso *ifEntry*.
  - ***getState(…)***. Permite actualizar la tabla entera de valores *ifEntry* de la tabla a través del Administrador Virtual de Hardware, e indica si hubo cambios de estado en la tabla entera.

- ***LIfEntry***. Clase que extiende de *ManagedObject*, la cual implementa a una entrada de la tabla *ifTable* de SNMP. Los métodos que se concretizan de la clase *ManagedObject* son:
  - ***setValue(…)***. Actualiza completa o parcialmente una entrada de la tabla *ifTable*.
  - ***getValue(…)***. Obtiene parte o totalmente una entrada de la tabla *ifTable*.
  - ***getState(…)***. Actualiza a través del Administrador Virtual de Hardware una entrada de la tabla *ifTable* e indica si hubo cambios de estado en la entrada actual.

8.5.3 Capa AVH

La funcionalidad de la aplicación en su capa MIB se encuentra definida en las siguientes secciones:

1. *Administración Virtual de Harware*. En esta sección se define en su totalidad la funcionalidad de la Administración Virtual de Hardware aplicada al ruteador. En ésta se describe como se monitorea el estado de interfaces seriales y ethernet. También se describe la forma en cómo se reportan los cambios de estado en éstos recursos.

8.5.3.1 Administración Virtual de Hardware

La Figura 8.11 muestra todas las clases de la aplicación utilizadas para proporcionar la fincionalidad de la Administración Virtual de Hardware. Estas clases son *RouterResourcesContainer* y *TrapReportCreator*, *RouterInterface*, y *RandomResourcesIterator*.

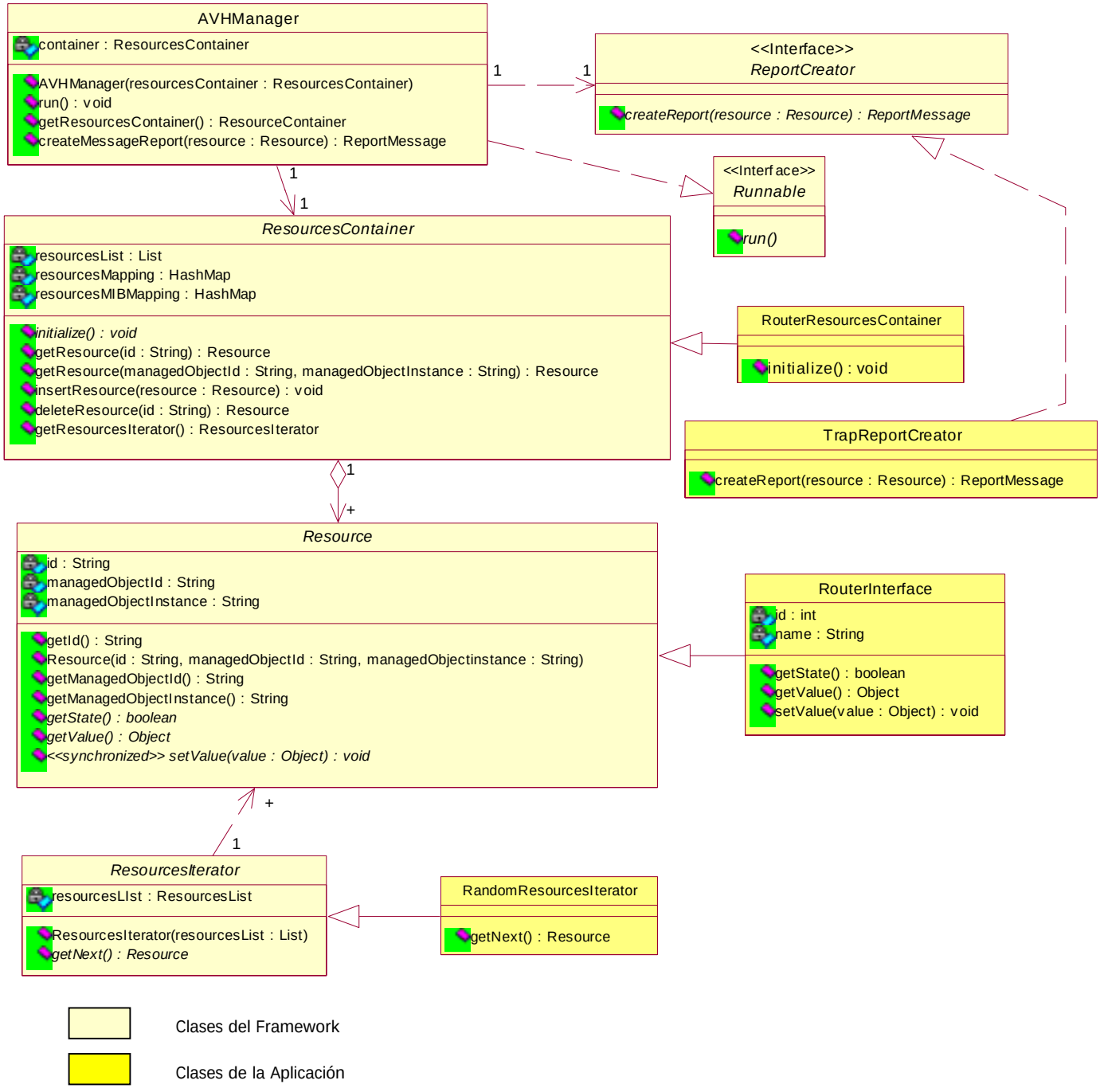


Figura 8.10 Administración Virtual de Hardware implementada en la aplicación.

Las clases implementadas en la aplicación son:

- ***RouterResourcesContainer***. Clase que extiende de *ResourcesContainer*, la cual inicializa los valores de los recursos del ruteador. Los métodos que se concretizan de la clase *ResourcesContainer* son:

- ***initialize(…)***. Obtiene e inicializa el estado de las interfaces seriales, ethernet y tablas ARP del ruteador.

- ***RandomResourcesIterator***. Clase que extiende de *ResourcesIterator*, la cual implementa el mecanismo de monitoreo de los recursos del ruteador. Los métodos que se concretizan de la clase *ResourcesIterator* son:
  - ***getNext(…)***. Obtiene el siguiente recurso del ruteador a monitorear. En este caso lo realiza de manera pseudoaleatoria.

- ***TrapReportCreator***. Clase que extiende de *ReportCreator*, la cual implementa el mecanismo de creación de reportes de cambio de estado en los recursos del ruteador. Los métodos que se concretizan de la clase *ReportCreator* son:
  - ***createReport(…)***. Genera un reporte de cambio de estado en cualquiera de los recursos del ruteador.

- ***RouterInterface***. Clase que extiende de *Resource*, la cual implementa la forma en que se obtiene el estado de las interfaces Ethernet0, Serial0 y Serial1 del ruteador. Los métodos que se concretizan de la clase *Resource* son:
  - ***getState(…)***. Obtiene el estado de cualquiera de las interfaces Ethernet0, Serial 0 y Serial 1 del ruteador, e indica si hubo un cambio de estado en alguna de éstas.

8.6 Implementación

8.6.1 Capa Agente

La capa Agente de la aplicación se compone de las siguientes clases:

- *CommunityFilter*. Filtra mensajes GET-REQUEST de acuerdo a la comunidad del Agente. Si lagún mensaje no pertenece a la comunidad del Agente, este es rechazado.
- *EventsFilter*. Filtra mensajes TRAP permitiendo solamente enviar mensajes al Administrador si no existe un enlace con alguna de las interfaces del ruteador.
- *FifoMessageQueue*. Implementa el mecanismo de contención de mensajes FIFO (First-In, First-Out).
- *GetService*. Implementación del Servicio GET-REQUEST del protocolo SNMP. Permite obtener objetos administrados de la MIB dada una lista de variables que discrimina la búsqueda.
- *PoolProcessManager*. Implementa el administrador de procesos del Agente. En este caso se implementa un mecanismo de "pool" de procesos para atender los servicios solicitados por el Administrador, o en su defecto por reportes de cambio de estado en los Recursos.
- *SNMPConstants*. Constantes SNMP para la interpretación y análisis de mensajes.
- *SNMPGetMessage*. Clase que implementa los mensajes SNMP GET-REQUEST y GET-RESPONSE.
- *SNMPMessageCoder*. Codificador de mensajes SNMP. En este caso codifica mensajes GET-RESPONSE y TRAP.
- *SNMPMessageDecoder*. Decodificador de mensajes SNMP. En este caso convierte un flujo de datos de entrada en mensajes GET-REQUEST.
- *SNMPObject*. Clase que contiene información referente a una variable SNMP, la cual contiene el tipo de dato, la logitud y el valor de la variable.
- *SNMPTrapMessage*. Clase que implementa mensajes TRAP de SNMP.
- *TrapService*. Implementación del Servicio TRAP del protocolo SNMP. Permite reportar cambios de estado en los Recursos del elemento de red.
- *UDPMessageReceptor*. Receptor de mensajes SNMP, en este caso implementa el mecanismo de datagramas para la recepción de mensajes.
- *UDPMessageTransmisor*. Transmisor de mensajes SNMP, en este caso implementa el mecanismo de datagramas para la emisión de mensajes.

Para verificar la implementación de éstas clases véase el "*Anexo D. Implementación del Agente Proxy SNMP para el Ruteador Cisco 2501*" en su sección "*D.2 Implementación de la Capa Agente*".

8.6.2 Capa MIB

La capa MIB de la aplicación se compone de las siguientes clases:

- *LIfEntry*. Clase que representa a un elemento de la tabla LIfTable de SNMP, el cual permite contener información del estado de interaces del ruteador Cisco 2501.
- *MIBConstants*. Constantes utilizadas en la MIB.
- *MIBDBContainer*. Clase que implementa el contenedor de Objetos Administrados de la MIB. En esta caso se implementa un contenedor de Base de Datos, el cual utiliza el puente JDBC-ODBC para conectarse con la BD en Access.

Para verificar la implementación de éstas clases véase el "*Anexo D. Implementación del Agente Proxy SNMP para el Ruteador Cisco 2501*" en su sección "*D.3 Implementación de la Capa MIB*".

8.6.3 Capa AVH

La capa AVH de la aplicación se compone de las siguientes clases:

- *AVHConstants*. Constantes utilizadas en la AVH.
- *RandomResourcesIterator*. Clase que implementa el mecanismo de iteración de Recursos del elemento de red, el cual es utilizado por el AVHManager para monitorear dichos Recursos.
- *RouterInterface*. Clase que representa el estado de las interfaces seriales y Ethernet del ruteador Cisco 2501.
- *RouterResourcesContainer*. Clase que implementa el contenedor de recursos de la AVH. En este caso se encarga de crear los tres tipos de interfaces del ruteador, así como de enviar el cambio de estado en el cual se indica que el Agente se ha inicializado.
- *StatusPeer*. Interfaz utilizada para conocer el estado de una conexión Telnet.
- *TelnetIO*. Clase que implementa el mecanismo entrada/salida de información a través de Telnet.
- *TelnetManager*. Clase que permite conectarse al ruteador Cisco 2501 a través de Telnet.
- *TelnetWrapper*. Clase que encapsula los mecanismo de envío y recepción de datos a través de Telnet.
- *TimedOutException*. Excepción de time-out en una conexión de Telnet.
- *TrapReportCreator*. Clase que permite crear reportes de cambio de estado en los Recursos del elemento de red.

Para verificar la implementación de éstas clases véase el "*Anexo D. Implementación del Agente Proxy SNMP para el Ruteador Cisco 2501*" en su sección "*D.4 Implementación de la Capa AVH*".

8.6.4 Archivo XML de Configuración

El archivo de configuración de la aplicación Agente es el siguiente:

```
<network-element id="NET09AK783909900"
                 ip="200.34.38.1"
                 name="Ruteador CISCO 2501"
                 description="Ruteador CISCO 2501">
   <agent>
      <agent-mandatory-resources>
         <message-decoder
          class="com.networkagent.ciscorouter.agent.SNMPMessageDecoder"/>
         <message-receptor
          class="com.networkagent.ciscorouter.agent.UDPMessageReceptor" port="161"/>
         <agent-filter-chain>
            <filter             class="com.networkagent.ciscorouter.agent.CommunityFilter"/>
            <filter             class="com.networkagent.ciscorouter.agent.EventsFilter"/>
         </agent-filter-chain>
         <agent-queue           class="com.networkagent.ciscorouter.agent.FifoMessageQueue"
                                capacity="50"/>
         <process-manager
          class="com.networkagent.networkelement.agent.PoolProcessManager" capacity="10"/>
         <services>
            <service            command-id="0xa0"
                                command-text="GET-REQUEST"
                                class="com.networkagent.ciscorouter.agent.GetService"/>
            <service            command-id="0xa4"
                                command-text="TRAP"
                                class="com.networkagent.ciscorouter.agent.TrapService"/>
         </services>
         <message-transmisor
         class="com.networkagent.networkelement.agent.UDPMessageTransmisor"/>
         <message-coder
         class="com.networkagent.networkelement.agent.SNMPMessageCoder"/>
         <administrators>
            <administrator      ip="200.34.38.1"  port="164"/>
         </administrators>
      </agent-mandatory-resources>
      <agent-additional-resources/>
      <agent-additional-classes/>
   </agent>
   <mib>
      <mib-mandatory-resources>
         <mib-container         class="com.networkagent.ciscorouter.mib.MIBDBContainer"/>
         <mib-filter-chain/>
         <mib-queue             class="com.networkagent.ciscorouter.agent.FifoMessageQueue"
                                capacity="30"/>
      </mib-mandatory-resources>
      <mib-additional-resources/>
      <mib-additional-classes/>
   </mib>
   <avh>
      <avh-mandatory-resources>
         <avh-manager           sleep-time="4000"/>
         <resources-container
          class="com.networkagent.ciscorouter.avh.RouterResourcesContainer"/>
         <resources-iterator
          class="com.networkagent.ciscorouter.avh.RandomResourcesIterator"/>
         <report-creator        class="com.networkagent.ciscorouter.avh.TrapReportCreator"/>
      </avh-mandatory-resources>
      <avh-additional-resources>
         <resource key   = "TelnetManager"
                   class = "com.networkagent.ciscorouter.avh.TelnetManager">
      </avh-additional-resources>
      <avh-additional-classes/>
   </avh>
</network-element>
```

En el archivo de configuración puede observarse como se implementaron los mecanismos de decodificación y codificación de mensajes respetando las reglas impuestas por SNMP, esto en las clases "*com.networkagent.ciscorouter.agent.SNMPMessage Decoder*" y "*com.networkagent.ciscorouter.agent.SNMPMessageCoder* " respectivamente.

A través de la utilización de UDPs, se implementa el mecanismo de recepción y transmisión de mensajes, esto definido en las clases " *com.networkagent.ciscorouter.agent. UDPMessageReceptor*" y "*com.networkagent.ciscorouter.agent.UDPMessageTransmisor*" respectivamente.

Para filtrar mensajes provenientes del Administrador y de la AVH, se encuentran definidos los filtros "*com.networkagent.ciscorouter.agent.CommunityFilter*" y "*com. networkagent.ciscorouter.agent.EventsFilter*".

Respecto al administrador de procesos, este mecanismo se define en la clase "*com.networkagent.ciscorouter.agent.PoolProcessManager*". Cabe destacar que la cola de mensajes de espera tanto del Agente como de la MIB se encuentra definida en la clase "*com. networkagent.ciscorouter.agent.FifoQueue*".

En la aplicación se definen dos servicios, el servicio GET-REQUEST y el servicio TRAP, los cuales se encuentran definidos en las clases "*com.networkagent.ciscorouter. agent.GetService*" y "*com.networkagent.ciscorouter. agent.TrapService*" respectivamente.

Respecto a la MIB, se define como contenedor de Objetos Administrados a "*com.networkagent.ciscorouter.mib.MIBDBContainer*" el cual se conecta a la Base de Datos de Access que almacena a los mismos.

Por último en la AVH se define el contenedor de recursos en la clase "*com.networkagent.ciscorouter. avh.RouterResourcesContainer*". El iterador de recursos se encuentra definido en la clase "*com.networkagent.ciscorouter.avh. RandomResourcesIterator*". El generador de reportes se encuentra definido en "*com.networkagent.ciscorouter.avh. TrapReportCreator*".

Cabe destacar que en la AVH se configuró un recurso adicional, el cual es un administrador de conexiones Telnet, esto con el fin de dejarlo disponible para su uso cuando los recursos del elemento de red necesiten obtener su estado. Este administrador de telnet se encuentra definido en "*com.networkagent.ciscorouter.avh. TelnetManager*".

Capítulo

9

Conclusiones Generales

9.1 Conclusiones del Desarrollo del Framework de Dominio

Dada mi perspectiva y gracias a la experiencia obtenida en la presente tesis, concluyo que el proceso de desarrollo de un framework de dominio debe ser el siguiente:

1. *Definición del Ámbito del Dominio*. En este caso se deben plasmar claramente las metas, objetivos y alcance del framework, así como los subdominios que serán involucrados en el mismo.
2. *Volverse un Experto del Dominio o involucrar a uno, en el desarrollo del framework.* En caso de que no se cuente con un experto de dominio, el cual proporcione los requerimientos necesarios para abstraer el dominio, es necesario realizar mucha investigación sobre el dominio, así como el analizar aplicaciones ya desarrolladas, esto con el fin de abstraer funcionalidad, conceptos y eventos que se manejen dentro del ámbito del dominio.
3. *Generar los Casos de Uso del Dominio*. En este caso lo primero que debe realizarse es obtener los casos de uso de cada uno de los subdominios, abstrerlos entre sí, para después abstraer los casos de uso resultantes y de esta forma obtener los casos de uso del dominio.
4. *Generar el Modelo Estático y Dinámico del Dominio.* Mediante el modelo estático y dinámico se pueden observar los conceptos que están involucrados en el dominio, así como sus relaciones e interacciones entre sí.
5. *Realizar un Análisis de Variabilidad*. El análisis de variabilidad es muy importante para poder visualizar las diferentes formas en que un concepto puede ser diseñado y hasta implementado, por lo cual se puede decir que ésta es una fase muy importante en el desarrollo de un framework.
6. *Diseñar el Framework utilizando Patrones de Diseño*. Es recomendable que el diseño del framework se oriente al uso de patrones de diseño debido a que son diseños ya probados e implementados en muchas aplicaciones, los cuales promueven el reuso del software.

7. *Implementar el Framework de Dominio*. En esta etapa se debe elegir un lenguaje de programación orientado a objetos que cuente con las bases suficientes como para implementar de manera efectiva el framework.
8. *Implementar Aplicaciones que usen el Framework*. Este punto es la fase culminante del proceso de desarrollo de un framework, debido a que además de validar el análisis y diseño del mismo, permite evolucionarlo y adaptarlo mejor a los objetivos y funcionalidad presente en las aplicaciones del dominio.

Respecto a la eficiencia del framework de dominio desarrollado existe una métrica que para tal caso. Para esto, [Fayad 2, 1999] realizó una encuesta entre desarrolladores de 41 frameworks diferentes, en la cual tuvo como resultado que el porcentaje de las clases que deben ser implementadas en una aplicación final varía entre un 10 % (Framework eficiente) y un 90 % (Framework deficiente). Dada esta panorámica a continuación se presenta una tabla comparativa entre las clases implementadas por el framework así como las clases implementadas en la aplicación.

Capa / Paquete	Clases del Framework		Clases de la Aplicación		Total	
	No.	%	No.	%	No.	%
Agente	15	25.00 %	14	23.33 %	**29**	**58.33 %**
MIB	3	5.00 %	4	6.66 %	**7**	**11.66 %**
AVH	5	8.34 %	11	18.33 %	**16**	**26.67 %**
Configuración	3	5.00 %	0	0.00 %	**3**	**5.00 %**
Utilerías	5	8.34 %	0	0.00 %	**5**	**8.34 %**
Total	**31**	**51.68 %**	**29**	**48.32 %**	**60**	**100.00 %**

Tabla 9.1 Eficiencia del Framework de Dominio.

A simple vista pareciera que el framework desarrollado mantiene un nivel medio de eficiencia (aproximadamente un 50 %), no obstante este nivel de eficiencia puede aumentar ya que se recomienda que varias de las clases desarrolladas por la aplicación se usen en el desarrollo de otras aplicaciones, como puede ser el caso de los servicios Get-Request y Trap ya implementados (solo en el caso de generar Agentes SNMP), el codificador y decodificador de mensajes, los filtros, el iterador de recursos, etc., no olvidando que uno de los principales objetivos del framework es su flexibilidad debido a que abarca dos subdominios que aunque son semejantes, tienen varias diferencias de funcionalidad, además que otro objetivo primordial es la extensibilidad de comandos del protocolo de administración de red a utilizar en los Agentes.

Por último, concluyo que el framework desarrollado corresponde de acuerdo a su ámbito, como un *Framework de Middleware de Integración*, debido a que permite implementar protocolos de administración de red para que los Agentes puedan comunicarse con los Administradores. De acuerdo a su técnica de extesión, se cataloga como un *Framework de Caja Gris*, debido a que utiliza para su extensibilidad tanto la herencia como el uso de interfaces. De acuerdo al tipo de información de administración que puede ser intercambiada entre los Agentes y Administradores el framework maneja los niveles *Elemental* y de *Administración de Elemento de Red*.

9.2 Trabajos Futuros

Partiendo de los resultados de la presente tesis, recomiendo el que se realicen los siguientes trabajos:

- ***Formalización del Método de Abstracción de Casos de Uso***. Debido a que en la tesis se presenta un método de abstracción de casos de uso, este puediera ser formalizado de tal forma que pudiera ser aceptado y publicado por la comunidad investigadora como un método válido para la abstracción de casos de uso y obtención de requerimientos de frameworks que involucran en su ámbito a más de dos subdominios.
- ***Transformación del Framework de Dominio de Caja Gris a Caja Negra***. Los frameworks de caja negra son mejor aceptados por los desarrolladores, ya que solo tienen que interpretar los contratos (interfaces) lo cual significa que no necesitan analizar código ya implementado para extender el framework. Debido a que el framework desarrollado es de caja gris, éste puede convertirse en uno de caja negra, lo cual pudiera aumentar su eficiencia y disminuir el número de clases que deben implementarse en nuevas aplicaciones.
- ***Generación de un Framework de Dominio para el desarrollo de Administradores de Red***. Un buen tema de tesis puede ser el desarrollar un Framework de Dominio para Administradores de Red (La contraparte de los Agentes). Ya que en la presente tesis existe el framework Agente, lo ideal sería que una vez desarrollado el framework Administrador, aplicaciones derivadas de los mismos interactúen y se comuniquen a través de un mismo protocolo de administración de red.
- ***Generación de Frameworks de Dominio para manejar niveles más altos de Información de Administración***. Un tema de tesis que puede ser la continuidad de mi tesis, es el desarrollar escalonadamente frameworks que permitan obtener información de administración de más alto nivel, empezando por el nivel deAdministración de Red, para continuar con el de Administración de Servicio y finalizar con el de Administración de Negocio.
- ***Generación de un Objeto de Negocio de Administración de Red***. Tal vez el proyecto más ambicioso es el desarrollar un *Objeto de Negocio*[12] *de Administración de Red*, el cual haga interactuar un conjunto de frameworks para poder obtener información de administración en sus distintos niveles de abstracción[13], lo que permitiría desarrollar una gama muy extensa de aplicaciones de administración de red.

[12] A un objeto de negocio se le denomina como un framework de frameworks.

[13] Recordemos que los niveles de abstracción en la información son: Elemental, Administración de Elemento de Red, Administración de Red, Administración de Servicio, Administración de Negocio.

9.3 Experiencia Adquirida

No quisiera concluir la redacción de la presente tesis sin antes exponer la experiencia personal adquirida en todo el desarrollo de la misma, desde su inicio hasta su término.

Como es lógico, el primer paso para desarrollar una tesis es el encontrar el tema que más se adapte a nuestros gustos, motivación y experiencia profesional. En mi caso el tema de tesis se dió gracias a la curiosidad que tuve cuando el Dr. Raúl Pérez mencionó que había aplicaciones llamadas "*frameworks*" los cuales utilizan el llamado Principio de Hollywood "*No nos llames, nosotros te llamaremos*", lo cual cambió totalmente el paradigma que tenía en aquel entonces del desarrollo de aplicaciones de software, las cuales siempre tenían el control del flujo de eventos.

Una vez que decidí aplicar el tema de los frameworks sobre mi tesis, debía encontrar entonces el dominio del framework, lo cual no fué difícil ya que estaba interesado en desarrollar aplicaciones de redes de telecomunicaciones, por lo que decidí enfocarme a los Agentes de Administración de Red, solo que en la funcionalidad de supervisión de alarmas.

Al principio la investigación se enfocó a conocer los conceptos utilizados por los frameworks, los tipos de frameworks, la forma de implementarlos, lo cual fué un proceso largo.

Posteriormente investigué respecto a la supervisión de alarmas en los Agentes de Administración de Red. Debido a que consideré que la supervisión de alarmas era un dominio muy pequeño, decidí que mi tesis se enfocaría a generar tres tipos de frameworks, los cuales serían de caja blanca, caja negra y caja gris, lo cual me di cuenta lamentablemente un semestre antes de terminar la maestría que era demasiado ambicioso, por lo cual decidí reorientar la tesis a las funcionalidades básicas presentes en los Agentes de Administración de Red.

La reorientación de la tesis permitió que ésta evolucionara sobre algo concreto y más factible, lo cual tuvo como resultado la presente tesis, no obstante, dado este hecho la misma no pudo ser concretada en el tiempo esperado (tres semestres). Aunque este hecho pudiera verse como una demora muy desafortunada, permitió que la misma madurara y se entregase a mi ver un trabajo con mucha mejor calidad.

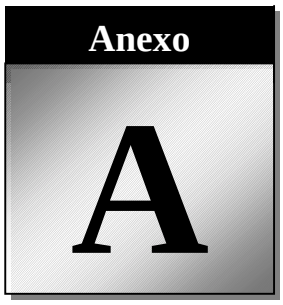
Como se puede observar, el desarrollo de la tesis fué un proceso difícil, pero satisfactorio, el cual permite observar el carácter y la forma en que cada uno de nosotros resuelve las adversidades que se presentan así también como tomamos decisiones importantes para poder desarrollar una tesis excelente.

Por lo mencionado con anterioridad, doy las siguientes sugerencias:

- El tener en mente áreas de interés, o mejor aún, un trabajo de tesis antes de cursar la maestría es un punto clave para poder terminarla en tiempo y forma, sobre todo si como en mi caso, son alumnos de tiempo completo.
- La dedicación a las materias de tesis debe ser igual o hasta más que las materias normales.
- La comunicación con el asesor respectivo debe ser clara, para que ambos sepan de antemano el resultado final esperado.
- El tema de tesis debe ser motivador para la persona que quiera desarrollarlo.

Por último, espero sinceramente que la presente tesis les sea de utilidad y de motivación ya que en ella está plasmada mi esfuerzo y dedicación para aportar un pequeño grano de arena en el ámbito de la investigación de tecnologías de información.

"*Un hombre fuerte no es aquel*
que puede levantar una montaña,
sino aquel que nunca se rinde".
Proverbio Chino.



Casos de Uso del Dominio

A.1 Introducción

En el presente anexo se describen los Casos de Uso pertenecientes tanto al modelo de información OSI e Internet. Estos casos de uso se obtuvieron principalmente a través de las fuentes de conocimiento [Udupa, 1999], [Black, 1992], y [NMF 037, 1995], en la cual se presentan las funcionalidades que un Agente debe tener para dar soporte a los comandos de SNMP y CMIP.

También se describen los casos abstractos de uso para cada modelo de información, los cuales son el resultado de aplicar la "*Abstracción Vertical*" en los mismos.

Por último, se presentan los casos abstractos del dominio, quienes son el resultado de aplicar la "*Abstracción Horizontal*" en el dominio, esto es, involucrando a los dos modelos de información.

A.2 Casos de Uso del Modelo OSI

Los casos de uso del modelo de información OSI describen la funcionalidad perteneciente a los Servicios Comunes de Información de Administración (CMIS de sus siglas en Inglés), los cuales tienen una correspondencia uno a uno con los comandos del protocolo de administración CMIP. Los servicios son los siguientes:

- M-EVENT-REPORT.
- M-GET.
- M-CANCEL-GET.
- M-SET.
- M-ACTION.
- M-CREATE.
- M-DELETE.

A continuación se describen los casos de uso para éstos servicios.

A.2.1 Caso de Uso del Servicio M-EVENT-REPORT

<table>
<tr><th colspan="2">Caso de uso:</th><th colspan="2">Reportar eventos generados en los Recursos</th></tr>
<tr><td colspan="2">Actores:</td><td colspan="2">Recurso(Iniciador), Agente CMIP(Sistema), Administrador CMIP.</td></tr>
<tr><td colspan="2">Propósito:</td><td colspan="2">Reportar al Administrador los cambios de estado que han experimentado los Recursos actualmente.</td></tr>
<tr><td colspan="2">Resumen:</td><td colspan="2">Un Recurso experimenta una falla o cambio de estado. El Objeto Administrado que representa al Recurso es notificado de esta situación a través de la AVH, genera la notificación correspondiente y se la hace llegar tanto a la Bitácora como al Discriminador de Envío de Eventos. Si la notificación cumple con los criterios de filtrado, ésta es convertida en un mensaje y es enviada al Administrador. En caso contrario, es desechada.</td></tr>
<tr><td colspan="2">Tipo:</td><td colspan="2">Primario y real.</td></tr>
<tr><td colspan="4">Sección: <u>Principal</u>
Curso normal de los eventos</td></tr>
<tr><td></td><td>Acción del actor</td><td></td><td>Respuesta del sistema</td></tr>
<tr><td>1.</td><td>Este caso de uso comienza cuando un Recurso experimenta una falla o un cambio de estado.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>2.</td><td>La Administración Virtual de Hardware detecta este evento y envía un reporte al Administrador de la MIB, en el cual contiene lo siguiente:
• Identificador del Recurso.
• Tipo de Evento.
• Información del Evento.</td></tr>
<tr><td></td><td></td><td>3.</td><td>El Administrador de la MIB relaciona el Recurso con su correspondiente Objeto Administrado (utilizando una lista que representa esta relación), y le envía el reporte generado.</td></tr>
<tr><td></td><td></td><td>4.</td><td>El Objeto Administrado analiza el reporte y en caso de que el evento sea una alarma, determina su severidad a través del Perfil de Asignación de Severidad de Alarmas.</td></tr>
<tr><td></td><td></td><td>5.</td><td>El Objeto Administrado genera una notificación, la cual contiene los siguientes parámetros:
• Identificador del Objeto Administrado.
• Tipo de Alarma.</td></tr>
</table>

			• Severidad Percibida. • Causa Probable. • Sugerencias de Solución.
		6.	El Objeto Administrado envía la notificación al Agente el cual la envía a las siguientes entidades: • Bitácora (Consúltese la sección "*Bitácora*"[14]). • Discriminador de Envío de Eventos (Consúltese la sección "*Discriminador de Envío de Eventos*"[1]).
		7.	Si la notificación no es desechada por el Discriminador, el Agente la recibe y genera un mensaje M-EVENT-REPORT con los siguientes parámetros: • Identificador de Invocación. • Modo. • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tipo de Evento. • Tiempo del Evento. • Información del Evento.
		8.	El Agente envía el mensaje al Administrador utilizando el protocolo CMIP y en caso de que el parámetro "Modo" haya sido de tipo "Confirmado", almacena en una lista el Identificador de Invocación del mensaje.
9.	El Administrador recibe el mensaje, lo analiza y le da el seguimiento correspondiente. **a)** En caso de que el parámetro "Modo" sea de tipo "*Confirmado*", consúltese la sección "*Respuesta al Evento*". **a)** En caso de que el parámetro "Modo" sea de tipo "No		

[14] Los procesos de Bitácora y del Discriminador de Envío de Eventos se efectúan concurrentemente, por lo cual son separados en secciones para indicar que no existe alguna secuencia entre ellos, es decir, no se puede saber a ciencia cierta que proceso se efectúa antes que el otro.

	Confirmado", no se genera ningún mensaje.		
Sección: *Bitácora*			
Curso normal de los eventos:			
	Acción del actor		**Respuesta del sistema**
		1.	El Filtro de Procesamiento de Bitácora determina si la notificación debe ser almacenada en la Bitácora, o debe ser desechada. Esto se determina por medio de criterios de filtrado, los cuales han sido preestablecidos en la configuración del Agente.
		2.	Si la notificación no es desechada, es almacenada en la Bitácora como un Objeto Administrado.
Sección: *Discriminador de Envío de Eventos*			
Curso normal de los eventos:			
	Acción del actor		**Respuesta del sistema**
		1.	El Discriminador de Envío de Eventos determina si la notificación debe enviarse al Administrador, o debe ser desechada. Esto se determina por medio de criterios de filtrado, los cuales han sido preestablecidos en la configuración del Agente. **a)** Si la notificación no es desechada, el Discriminador de Envío de Evento la envía al Agente. **b)** En caso contrario, la notificación es desechada.
Sección: *Respuesta al Evento*			
Curso normal de los eventos:			
	Acción del actor		**Respuesta del sistema**
1.	El Administrador genera un mensaje de tipo M-EVENT-REPORT con los siguientes parámetros: • Identificador de Invocación. • Clase del Objeto Administrado.		

2.	• Instancia del Objeto Administrado. • Tipo de Evento. • Tiempo Actual. • Respuesta al Evento. • Errores. El Administrador envía el mensaje al Agente mediante el protocolo CMIP.	**3**.	El Agente recibe el mensaje, determina el tipo de solicitud (que en este caso es de tipo M-EVENT-REPORT), y realiza una búsqueda del Identificador de Invocación en su lista de Identificadores para relacionarlo con un evento generado con anterioridad. **a)** En caso de que el identificador exista en la lista, se verifica el parámetro "Respuesta al Evento", y dependiendo de éste se realiza una acción. **b)** En caso contrario el mensaje es desechado.

A.2.2 Caso de Uso del Servicio M-GET

<table>
<tr><th colspan="2">Caso de uso:</th><th colspan="2">Obtener información de Objetos Administrados</th></tr>
<tr><td colspan="2">Actores:</td><td colspan="2">Administrador CMIP (Iniciador), Agente CMIP (Sistema).</td></tr>
<tr><td colspan="2">Propósito:</td><td colspan="2">Obtener los valores actuales de atributos de Objetos Administrados.</td></tr>
<tr><td colspan="2">Resumen:</td><td colspan="2">El Administrador desea obtener la información de algunos atributos de Objetos Administrados, por lo cual envía una solicitud al Agente, ésta es atendida, y en caso de que todos los atributos existan y cuenten con permisos de lectura, se le retorna al Administrador la información solicitada, en caso contrario se le retorna un mensaje de error.</td></tr>
<tr><td colspan="2">Tipo:</td><td colspan="2">Primario y real.</td></tr>
<tr><td colspan="4">Sección: <u>Principal</u>
Curso normal de los eventos</td></tr>
<tr><td></td><td>Acción del actor</td><td></td><td>Respuesta del sistema</td></tr>
<tr><td>1.</td><td>Este caso de uso comienza cuando el Administrador desea solicitar información sobre uno o varios atributos de Objetos Administrados.</td><td></td><td></td></tr>
<tr><td>2.</td><td>El Administrador genera un mensaje M-GET con los siguientes parámetros:
• Identificador de Invocación.
• Clase del Objeto Base.
• Instancia del Objeto Base.
• Ámbito.
• Filtro.
• Control de Acceso.
• Sincronización.
• Lista de Identificadores de Atributos.</td><td></td><td></td></tr>
<tr><td>3.</td><td>El Administrador envía el mensaje al Agente mediante el protocolo CMIP.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>4.</td><td>El Agente recibe el mensaje, determina el tipo de solicitud, la almacena en una lista de espera y guarda también el Identificador de Invocación en otra lista.</td></tr>
<tr><td></td><td></td><td>5.</td><td>Una vez que el Agente puede procesar la</td></tr>
</table>

			solicitud, éste le envía varios parámetros al Administrador de la MIB para que realice la búsqueda de los Objetos Administrados a los cuales pertenecen cada uno de los Atributos definidos en la solicitud. Estos parámetros son: • Clase del Objeto Base. • Instancia del Objeto Base. • Lista de Identificadores de Atributos. • Ámbito. • Filtro. • Control de Acceso.
		6.	El Administrador de la MIB realiza la búsqueda de los Objetos Administrados. **a)** En caso de que existan, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que al menos uno no exista, se genera una notificación para cada Objeto no existente, con los siguientes parámetros: • Identificador de Invocación. • Identificador de Enlace (Identificador de Invocación del mensaje M-GET del Administrador). • Clase del Objeto Administrado (Clase del Objeto Administrado que no se encontró). • Instancia del Objeto Administrado. • Tiempo Actual. • Lista de Atributos (Nula). • Errores (Se informa la inexistencia del Objeto Administrado).
		7.	El Administrador de la MIB envía las notificaciones generadas al Agente.
		8.	El Agente recibe las notificaciones y crea un mensaje de tipo M-GET para cada una de ellas, el cual contiene los siguientes parámetros: • Identificador de Invocación. • Identificador de Enlace. • Clase del Objeto Administrado.

		9.	• Instancia del Objeto Administrado. • Tiempo Actual. • Lista de Atributos. • Errores. El Agente envía al Administrador cada uno de los mensajes, utilizando el protocolo CMIP y elimina el identificador de invocación de la lista.
10.	El Administrador recibe los mensajes, e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró todos los Objetos Administrados.
		2.	Para cada atributo solicitado, se analiza que éste exista y que tenga permisos de lectura. **a)** En caso de que todos los atributos existan y tengan permisos de lectura, se genera una notificación para cada grupo de atributos que pertenezca a un mismo Objeto Administrado. Esta notificación va a contener los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. **b)** En caso contrario, se genera una notificación para cada grupo de atributos que pertenezca a un mismo Objeto Administrado y que al menos un Atributo no tenga permisos de lectura. Esta notificación va a contener los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Errores (Se especifican los Atributos que no tienen permisos de lectura).

A.2.3 Caso de Uso del Servicio M-CANCEL-GET

Caso de uso:	**Cancelar una solicitud de información**
Actores:	Administrador CMIP (Iniciador), Agente CMIP (Sistema).
Propósito:	Cancelar una solicitud de información hecha con anterioridad por el Administrador.
Resumen:	El Administrador envía una solicitud de tipo M-CANCEL-GET al Agente para cancelar una solicitud M-GET realizada con anterioridad. El Agente la recibe, interpreta el estado actual de la solicitud M-GET, si aún esta siendo procesada o está pendiente, la cancela, si ya ha sido procesada genera un mensaje de error.
Tipo:	Primario y real.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea cancelar una solicitud de información (M-GET) realizada con anterioridad.		
2.	El Administrador almacena dentro de un mensaje de tipo M-CANCEL-GET los siguientes parámetros: • Identificador de Invocación. • Identificador de Invocación de GET (Identificador del mensaje M-GET el cual se envió con anterioridad).		
3.	El Administrador envía el mensaje al Agente mediante el protocolo CMIP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, y en este caso no la almacena en una lista de espera, sino que la atiende de forma casi inmediata.
		5.	El Agente realiza una búsqueda del Identificador de Invocación GET en su lista de Identificadores.

			a) Si se encontró el identificador, consúltese la sección "*Búsqueda Exitosa*". **b)** En caso contrario, el Agente genera un mensaje de tipo M-CANCEL-GET el cual va a contener los siguientes parámetros: • Identificador de Invocación. • Errores (Aquí se especifica que la solicitud M-GET no existe o ya fue procesada).
		6.	El Agente envía el mensaje generado al Administrador, utilizando el protocolo CMIP.
7.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Agente busca el Identificador de Invocación GET en su lista de espera. **a)** Si encuentra el identificador, elimina el mensaje de la lista de espera, y genera un mensaje de tipo M-CANCEL-GET, el cual va a contener los siguientes parámetros: • Identificador de Invocación. • Errores (Nulo). **b)** Si no se encuentra el identificador, el Agente cancela el procesamiento de la solicitud y genera un mensaje de tipo M-CANCEL-GET, el cual va contener los siguientes parámetros: • Identificador de Invocación. • Errores (Nulo).

A.2.4 Caso de Uso del Servicio M-SET

Caso de uso:	**Modificar información de Objetos Administrados**
Actores:	Administrador CMIP (Iniciador), Agente CMIP (Sistema).
Propósito:	Modificar los valores de uno o varios atributos de Objetos Administrados.
Resumen:	El Administrador desea modificar la información de algunos atributos de Objetos Administrados, por lo cual envía una solicitud al Agente, ésta es atendida, y en caso de que todos los atributos existan y de que se cuenten con los permisos de escritura, se le retorna al Administrador la información que ha sido modificada, en caso contrario se le retorna un mensaje de error.
Tipo:	Primario y real.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea modificar información sobre uno o varios atributos de Objetos Administrados.		
2.	El Administrador almacena dentro de un mensaje de tipo M-SET los siguientes parámetros: • Identificador de Invocación. • Modo. • Clase del Objeto Base. • Instancia del Objeto Base. • Ámbito. • Filtro. • Control de Acceso. • Sincronización. • Lista de Atributos.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo CMIP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, la almacena en una lista de espera y guarda también el Identificador de Invocación en otra lista.
		5.	Una vez que el Agente puede procesar la

			solicitud, éste le envía varios parámetros al Administrador de la MIB para que realice una búsqueda de los Objetos Administrados a los cuales pertenece cada uno de los Atributos, a partir del Objeto Administrado Base. Estos parámetros son: • Clase del Objeto Base. • Instancia del Objeto Base. • Lista de Identificadores de Atributos. • Ámbito. • Filtro. • Control de Acceso.
		6.	El Administrador de la MIB realiza la búsqueda. **a)** En caso de que todos los Atributos existan, consúltese la sección "*Búsqueda Exitosa*". **b)** En caso de que al menos un Atributo no exista, se genera una notificación para el grupo de Atributos no existentes. Esta notificación va a contener los siguientes parámetros: • Clase del Objeto Administrado (Nulo). • Instancia del Objeto Administrado (Nula). • Lista de Atributos. • Errores (En el se especifica que los Atributos de la lista no existen).
		7.	El Administrador de la MIB envía las notificaciones generadas al Agente.
		8.	El Agente recibe las notificaciones y por cada una de ellas genera un mensaje de tipo M-SET, con lo siguiente: • Identificador de Invocación. • Identificador de Enlace (Identificador de Invocación del mensaje M-SET del Administrador.). • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tiempo Actual. • Lista de Atributos.

		9.	• Errores. El Agente envía cada uno de los mensajes al Administrador, utilizando el protocolo CMIP.
10.	El Administrador recibe los mensajes, interpreta su información y le da el seguimiento correspondiente.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	Para cada atributo solicitado, se verifica que cuente con permisos de escritura. **a)** En caso de que todos los atributos cuenten con permisos de escritura, se cambian sus valores por los correspondientes en la solicitud. Se genera además una notificación por cada grupo de atributos que pertenezcan a un mismo Objeto Administrado. Cada una de las notificaciones van a contener lo siguiente: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. **b)** En caso contrario, se genera una notificación por cada grupo de Atributos que pertenezcan a un mismo Objeto Administrado y que no cuenten con permisos de escritura. Las notificaciones van a contener lo siguiente: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Errores (Se especifica que los Atributos de la lista no cuentan con permisos de lectura.).
		2.	Después de que se modificaron atributos en Objetos Administrados, estos cambios se deben de reflejar en los Recursos según sea el caso (Véase la sección

			"*Modificación de atributos en los Recursos*".).
Sección: *Modificación de atributos en los Recursos* **Curso normal de los eventos**			
	Acción del actor		**Respuesta del sistema**
		1.	Se busca el Recurso que está relacionado con el Objeto Administrado mediante una lista que representa esta relación. **a)** En caso de que no exista tal relación no se realiza ninguna acción. **b)** En caso de que exista la relación, se le envía a la Administración Virtual de Hardware un reporte en el que se especifica que hubo un cambio de valor en un Recurso. El Reporte va a contener los siguientes parámetros: • Identificador del Recurso. • Identificador de la Acción. • Lista de Atributos a modificar.
		2.	La Administración Virtual de Hardware realiza una búsqueda del Recurso. **c)** En caso de que no exista, se genera un reporte de falla al Administrador de la MIB (Véase el Caso de Uso "*Reportar alarmas generadas en los Recursos*".). **d)** En caso de que exista se cambian los valores del Recurso que correspondan.

A.2.5 Caso de Uso del Servicio M-ACTION

Caso de uso:	**Invocar servicios en un Objeto Administrado**
Actores:	Administrador CMIP (Iniciador), Agente CMIP (Sistema).
Propósito:	Ejecución de un servicio remoto en Objetos Administrados.
Resumen:	El Administrador desea que un Objeto Administrado ejecute un servicio remoto, por lo cual le envía una solicitud al Agente, éste la interpreta, se realiza una búsqueda del Objeto Administrado, en caso de que el servicio pueda ser ejecutado se le retorna al Administrador la información resultante, en caso contrario se retorna un mensaje de error.
Tipo:	Primario y real.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea ejecutar un servicio remoto en uno de los Objetos Administrados en el ámbito del Agente.		
2.	El Administrador almacena dentro de un mensaje de tipo M-ACTION los siguientes parámetros: • Identificador de Invocación. • Modo (Conf./No Conf.). • Clase del Objeto Base. • Instancia del Objeto Base. • Ámbito. • Filtro. • Control de Acceso. • Sincronización. • Tipo de Acción. • Argumento de la Acción.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo CMIP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, la almacena en una lista de espera y guarda también el Identificador de Invocación en otra lista.

		5.	Una vez que el Agente puede procesar la solicitud, éste le envía varios parámetros al Administrador de la MIB, con estos parámetros se busca el primer Objeto Administrado que puede realizar la acción solicitada. La búsqueda se realiza a partir del Objeto Administrado Base, utilizando como criterio el parámetro "Filtro". Los parámetros son: • Clase del Objeto Base. • Instancia del Objeto Base. • Filtro. • Control de Acceso. • Tipo de Acción. • Argumento de la Acción.
		6.	El Administrador de la MIB realiza la búsqueda. **a)** En caso de que se encuentre a un Objeto Administrado que pueda realizar la acción, consúltese la sección "*Búsqueda Exitosa*". **b)** En caso de que no se encuentre ningún Objeto Administrado que pueda realizar la acción, se genera una notificación la cual contiene los siguientes parámetros: • Tipo de Acción. • Resultado de la Acción (Nula.). • Errores (Se especifica que no se encontró ningún Objeto que pueda realizar la acción.).
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación, obtiene de la lista el Identificador de Invocación de la solicitud del Administrador, y genera un mensaje de tipo M-ACTION con los siguientes parámetros: • Identificador de Invocación. • Identificador de Enlace. • Clase del Objeto Administrado. • Instancia del Objeto Administrado.

			• Tipo de Acción. • Tiempo Actual. • Resultado de la Acción. • Errores.
10.	El Administrador recibe el mensaje, interpreta su información y le da el seguimiento correspondiente.	**9**.	El Agente envía el mensaje al Administrador utilizando el protocolo CMIP.
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB envía al Objeto Administrado los parámetros que el Administrador proporcionó para la ejecución del servicio.
		2.	El Objeto Administrado interpreta los parámetros. **a)** En caso de que el servicio solicitado pueda ser ejecutado por el Objeto Administrado y que los parámetros sean los adecuados, se ejecuta el servicio (Véase *Paso 3*.). **b)** En caso de que el servicio solicitado no pueda ser ejecutado debido a que no se cuentan con los permisos necesarios (Parámetro “Control de Acceso”.), se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tipo de Acción. • Resultado de la Acción (Nulo.). • Errores (Se especifica que no se cuenta con los permisos necesarios para efectuar el servicio) **c)** En caso de que el servicio solicitado no pueda ser ejecutado debido a que el “Argumento de la Acción” no corresponda con el servicio, se genera una notificación con los siguientes

			parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tipo de Acción. • Resultado de la Acción (Nulo.). • Errores (Se especifica que el servicio no puede ser ejecutado debido a que los argumentos no corresponden con el servicio.).
		3.	Si se da el caso, se ejecuta el servicio. **a)** Si al ejecutar el servicio, surgen problemáticas, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tipo de Acción. • Resultado de la Acción (Nulo.). • Errores (Se especifican los errores que surgieron al ejecutarse el servicio.). **b)** En caso de que se ejecute eficazmente el servicio, se genera un mensaje, con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tipo de Acción. • Resultado de la Acción. • Errores (Nulo.).
		4.	La notificación generada es enviada al Administrador de la MIB.

A.2.6 Caso de Uso del Servicio M-CREATE

<table>
<tr><th>Caso de uso:</th><th colspan="3">Crear un Objeto Administrado</th></tr>
<tr><td>Actores:</td><td colspan="3">Administrador CMIP (Iniciador), Agente CMIP (Sistema).</td></tr>
<tr><td>Propósito:</td><td colspan="3">La creación de un Objeto Administrado en el dominio del Agente.</td></tr>
<tr><td>Resumen:</td><td colspan="3">El Administrador desea crear un Objeto Administrado en la MIB del Agente, por lo cual le envía una solicitud, ésta es atendida, y en caso de que el Objeto Administrado pueda ser creado, se le retorna al Administrador un mensaje que contiene información sobre el objeto, en caso contrario se retorna un mensaje de error.</td></tr>
<tr><td>Tipo:</td><td colspan="3">Primario y real.</td></tr>
<tr><td colspan="4">Sección: <u>Principal</u>
Curso normal de los eventos</td></tr>
<tr><td></td><td>Acción del actor</td><td></td><td>Respuesta del sistema</td></tr>
<tr><td>1.</td><td>Este caso de uso comienza cuando el Administrador desea crear un Objeto Administrado en la MIB del Agente.</td><td></td><td></td></tr>
<tr><td>2.</td><td>El Administrador genera un mensaje de tipo M-CREATE con los siguientes parámetros:
• Identificador de Invocación.
• Clase del Objeto Administrado.
• Instancia del Objeto Administrado.
• Instancia del Objeto Superior.
• Control de Acceso.
• Instancia del Objeto Referencia.
• Lista de Atributos.</td><td></td><td></td></tr>
<tr><td>3.</td><td>El Administrador envía el mensaje al Agente, utilizando el protocolo CMIP.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>4.</td><td>El Agente recibe el mensaje, determina el tipo de solicitud, y la almacena en una lista de espera.</td></tr>
<tr><td></td><td></td><td>5.</td><td>Una vez que el Agente puede procesar la solicitud, éste le envía parámetros al Administrador de la MIB para que realice la búsqueda del Objeto Administrado</td></tr>
</table>

			"Superior". Los parámetros son: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Instancia del Objeto Superior. • Instancia del Objeto Referencia. • Control de Acceso. • Lista de Atributos.
		6.	El Administrador de la MIB realiza la búsqueda. **a)** En caso de que exista el Objeto Superior, consúltese la sección "*Búsqueda Exitosa*". **b)** En caso de que no se proporcione un Objeto Superior, se crea un Objeto Administrado dada la Clase de Objeto Administrado proporcionada. En caso de que se tenga un objeto referencia, se copian de éste todos sus atributos hacia el nuevo Objeto Administrado, o en caso de que se proporcionen los valores de los atributos del nuevo Objeto Administrado, éstos son copiados hacia el Objeto. **c)** En caso de que no exista, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Errores (En el se especifica que no existe el Objeto "Padre").
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación, y genera un mensaje de tipo M-CREATE con los siguientes parámetros: • Identificador de Invocación. • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Tiempo Actual.

		9.	• Errores. El Agente envía el mensaje al Administrador utilizando el protocolo CMIP.
10.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto padre y le envía los parámetros para crear un nuevo Objeto Administrado.
		2.	El objeto padre verifica la "Clase de Objeto Administrado" para asegurarse de que el Objeto Administrado a crear pertenece a una de las "Clases de Objetos Administrados" de sus hijos inmediatos. **a)** En caso de que alguna de ellas coincida, y de que se tengan los permisos necesarios para crear el nuevo objeto, éste es creado y almacenado en la MIB, en su lugar correspondiente. Además se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. **b)** En caso de que no se cuente con los permisos necesarios para crear el objeto, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Errores (Se especifica que no se cuentan con los permisos necesarios para crear el Objeto). **c)** En caso de que la Clase del Objeto Administrado no coincida con ninguna que pertenezca a los hijos del objeto

			padre, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Errores (Se especifica que no coincide la Clase del Objeto "nuevo" con ninguna Clase de los Objetos "hijo").

A.2.7 Caso de Uso del Servicio M-DELETE

<table>
<tr><td>Caso de uso:</td><td colspan="3">Borrar un Objeto Administrado</td></tr>
<tr><td>Actores:</td><td colspan="3">Administrador CMIP (Iniciador), Agente CMIP (Sistema).</td></tr>
<tr><td>Propósito:</td><td colspan="3">Borrar un Objeto Administrado en el dominio del Agente.</td></tr>
<tr><td>Resumen:</td><td colspan="3">El Administrador desea borrar un Objeto Administrado en la MIB del Agente, por lo cual le envía una solicitud, ésta es atendida, y en caso de que el Objeto Administrado pueda ser borrado, se le retorna al Administrador un mensaje el cual contiene información sobre esta acción, en caso contrario se le retorna un mensaje de error.</td></tr>
<tr><td>Tipo:</td><td colspan="3">Primario y real.</td></tr>
<tr><td colspan="4">Sección: <u>Principal</u>
Curso normal de los eventos</td></tr>
<tr><td></td><td>Acción del actor</td><td></td><td>Respuesta del sistema</td></tr>
<tr><td>1.</td><td>Este caso de uso comienza cuando el Administrador desea borrar uno o más Objetos Administrados en la MIB del Agente.</td><td></td><td></td></tr>
<tr><td>2.</td><td>El Administrador genera un mensaje de tipo M-DELETE con los siguientes parámetros:
• Identificador de Invocación.
• Clase del Objeto Base.
• Instancia del Objeto Base.
• Ambito.
• Filtro.
• Control de Acceso.
• Sincronización.</td><td></td><td></td></tr>
<tr><td>3.</td><td>El Administrador envía el mensaje al Agente, utilizando el protocolo CMIP.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>4.</td><td>El Agente recibe el mensaje, determina el tipo de solicitud, la almacena en una lista de espera y guarda también el Identificador de Invocación en otra lista.</td></tr>
<tr><td></td><td></td><td>5.</td><td>Una vez que el Agente puede procesar la solicitud, éste le envía los parámetros necesarios al Administrador de la MIB para que realice la búsqueda del Objeto Administrado Base. Los parámetros son:</td></tr>
</table>

			• Clase del Objeto Base. • Instancia del Objeto Base. • Filtro. • Control de Acceso.
		6.	El Administrador de la MIB realiza la búsqueda. **a)** En caso de que exista, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que no exista, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Lista de Atributos. • Errores (Se especifica que no fue encontrado el Objeto Base).
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación, obtiene de la lista el Identificador de Invocación de la solicitud del Administrador, y genera un mensaje de tipo M-DELETE con los siguientes parámetros: • Identificador de Invocación. • Identificador de Enlace. • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Tiempo Actual. • Errores.
		9.	El Agente envía el mensaje al Administrador utilizando el protocolo CMIP.
10.	El Administrador recibe el mensaje e interpreta su información.		

Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto Administrado Base.
		2.	Se realiza una búsqueda de los Objetos Administrados que se encuentren dentro del criterio de filtrado: **a)** En caso de que se encuentren Objetos Administrados que cumplan con el Filtro, se verifica a cada uno que cuenten con los permisos necesarios para su borrado. En caso afirmativo, se eliminan los Objetos Administrados y se genera una notificación para cada Objeto eliminado la cual contiene los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Errores (Nulo). En caso contrario, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado. • Instancia del Objeto Administrado. • Errores (Se especifica que no se cuenta con los permisos necesarios para borrar al Objeto Administrado). **b)** En caso de que no se encuentre a ningún Objeto Administrado que cumpla con los criterios de filtrado, se genera una notificación con los siguientes parámetros: • Clase del Objeto Administrado (Nulo). • Instancia del Objeto Administrado (Nulo). • Errores (Se especifica que no se encontró a ningún Objeto Administrado que cumpla con los criterios de filtrado).

A.3 Casos de Uso del Modelo de Internet

Los casos de uso del modelo de información de Internet describen principalmente la funcionalidad que el Agente debe brindar a los comandos del protocolo de administración SNMP. Los comandos son los siguientes:

- TRAP.
- GET-REQUEST.
- GET-NEXT-REQUEST.
- SET-REQUEST.

A continuación se describen los casos de uso para éstos comandos.

A.3.1 Caso de Uso del Servicio TRAP

Caso de uso:	**Reportar eventos generados en los Recursos**
Actores:	Recurso(iniciador), Agente SNMP(Sistema), Administrador SNMP.
Propósito:	Reportar al Administrador los cambios de estado que han experimentado los Recursos actualmente.
Resumen:	Un Recurso experimenta una falla o cambio de estado. La Administración Virtual reporta el cambio de estado al Administrador de la MIB, este relaciona al Recurso el Objeto Administrado correspondiente y modifica su valor de acuerdo al reporte. El objeto modificado le indica al Agente esta situación. El Agente verifica los valores frontera del objeto y si estos rebasan alguno, se genera un TRAP el cual es enviado al Administrador.
Tipo:	Primario y real.

Sección: *Principal*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando un Recurso experimenta una falla o cambio de estado.		
		2.	La Administración Virtual de Hardware detecta el evento y envía un reporte al Administrador de la MIB, en el cual contiene lo siguiente: • Identificador del Recurso. • Tipo de Falla. • Información de la Falla.
		3.	El Administrador de la MIB relaciona al Recurso con el Objeto Administrado correspondiente y modifica su valor.
		4.	Al modificarse el valor de un Objeto Administrado, este le envía una notificación al Agente de este suceso. Esta notificación contiene los siguientes parámetros: • Indice del Objeto Administrado. • Tipo de Notificación (Cambio de estado.).
		5.	El Agente analiza este cambio de valor utilizando un esquema de filtrado. **a)** En caso de que el valor rebase los

			límites frontera se genera un mensaje de tipo TRAP, el cual contendrá los siguientes parámetros: • Identificador del PDU. • Indice del Objeto Administrado. • Dirección del Agente. • Trap Genérica. • Trap Específica (Se especifica el significado del cambio de estado). • Marca de Tiempo. • Lista de Variables de Información Adicional (En caso de que se amerite). **b)** En caso de que el valor no rebase los límites frontera, el Agente no realiza ninguna acción y se finaliza con este caso de uso.
		6.	El Agente envía el mensaje utilizando el protocolo SNMP.
7.	El Administrador recibe el mensaje y le da el seguimiento pertinente.		

A.3.2 Caso de Uso del Servicio GET-REQUEST

<table>
<tr><th colspan="2">Caso de uso:</th><th colspan="2">Obtener el valor de un Objeto Administrado</th></tr>
<tr><td colspan="2">Actores:</td><td colspan="2">Administrador SNMP (Iniciador), Agente SNMP (Sistema).</td></tr>
<tr><td colspan="2">Propósito:</td><td colspan="2">Obtener el valor actual de un Objeto Administrado.</td></tr>
<tr><td colspan="2">Resumen:</td><td colspan="2">El Administrador desea obtener información de un Objeto Administrado, por lo cual envía una solicitud al Agente, ésta es atendida, y en caso de que el Objeto Administrado exista y tenga permisos de lectura, se le retorna al Administrador la información solicitada, en caso contrario se le retorna un mensaje de error.</td></tr>
<tr><td colspan="2">Tipo:</td><td colspan="2">Primario y real.</td></tr>
<tr><td colspan="4">Sección: <u>Principal</u>
Curso normal de los eventos</td></tr>
<tr><td></td><td>Acción del actor</td><td></td><td>Respuesta del sistema</td></tr>
<tr><td>1.</td><td>Este caso de uso comienza cuando el Administrador desea solicitar información sobre un Objeto Administrado.</td><td></td><td></td></tr>
<tr><td>2.</td><td>El Administrador genera un mensaje GET-REQUEST con los siguientes parámetros:
• Identificador del PDU.
• Error de Estado.
• Indice de la Variable que generó el error.
• Nombre del Objeto Administrado.
• Valor del Objeto Administrado.</td><td></td><td></td></tr>
<tr><td>3.</td><td>El Administrador envía el mensaje al Agente mediante el protocolo SNMP.</td><td></td><td></td></tr>
<tr><td></td><td></td><td>4.</td><td>El Agente recibe el mensaje, determina el tipo de solicitud, y la almacena en una lista de espera.</td></tr>
<tr><td></td><td></td><td>5.</td><td>Una vez que el Agente puede procesar la solicitud, éste le envía los parámetros necesarios al Administrador de la MIB para que realice la búsqueda del Objeto Administrado. Los parámetros:
• Identificador del PDU.</td></tr>
</table>

			• Nombre del Objeto Administrado.
		6.	El Administrador de la MIB realiza la búsqueda del Objeto Administrado. **a)** En caso de que exista, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que no exista, se genera un mensaje de error.
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación y genera un mensaje de tipo GET-RESPONSE con los mismos parámetros que la notificación, pero le añade un identificador de solicitud.
		9.	El Agente envía el mensaje al Administrador utilizando el protocolo SNMP.
10.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto Administrado. **a)** En caso de que se cuenten con los permisos de lectura, se genera una notificación, la cual contiene el Objeto Administrado o el primer valor del mismo (en caso de que se trate de una colección de valores o tabla). La notificación tiene los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado. • Indice de la variable que generó el error (Nulo.). • Error de Estado (Nulo.). **b)** En caso contrario, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado.

			• Valor del Objeto Administrado (Nulo). • Indice de la variable que generó el error (Este parámetro tiene un valor sólo si el Objeto Administrado es una tabla.). • Error de Estado (El cual especifica que no se pudo obtener el Objeto Administrado por que no cuenta con permisos de lectura.).

A.3.3 Caso de Uso del Servicio GET-NEXT-REQUEST

Caso de uso:	**Obtener el valor de un Objeto Administrado**
Actores:	Administrador SNMP (Iniciador), Agente SNMP (Sistema).
Propósito:	Obtener el siguiente valor de un Objeto Administrado.
Resumen:	El Administrador desea obtener el siguiente valor de un Objeto Administrado, por lo cual envía una solicitud al Agente, ésta es atendida, y en el caso de que el Objeto Administrado exista, que represente una colección de valores, que tenga permisos de lectura y que con anterioridad se haya utilizado una solicitud de tipo GET-REQUEST o GET-NEXT-REQUEST, se le retorna al Administrador el siguiente valor del Objeto, en caso contrario se le retorna un mensaje de error.
Tipo:	Primario y real.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea solicitar el siguiente valor de un Objeto Administrado que representa una colección de valores.		
2.	El Administrador genera un mensaje de tipo GET-NEXT-REQUEST con los siguientes parámetros: • Identificador del PDU. • Error de Estado. • Indice del siguiente valor. • Nombre del Objeto Administrado. • Valor del Objeto Administrado.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo SNMP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, y la almacena en una lista de espera.
		5.	Una vez que el Agente puede procesar la

			solicitud, éste le envía los parámetros necesarios al Administrador de la MIB para que realice la búsqueda del Objeto Administrado. Los parámetros son los siguientes: • Identificador del PDU. • Nombre del Objeto Administrado. • Indice del siguiente valor.
		6.	El Administrador de la MIB realiza la búsqueda del Objeto Administrado. **a)** En caso de que exista, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que no exista, se genera un mensaje de error.
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación y genera un mensaje de tipo GET-RESPONSE con los mismos parámetros que la notificación, pero le añade un identificador de solicitud.
		9.	El Agente envía el mensaje al Administrador utilizando el protocolo SNMP.
10.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto Administrado.
		2.	El Administrador de la MIB intenta obtener el siguiente valor en la tabla. **a)** En caso de que se cuenten con los permisos de lectura y el índice de la variable no sobrepase el final de la tabla, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado.

			• Valor del Objeto Administrado. • Indice de la variable. **b)** En caso de que se cuenten con los permisos de lectura y el índice de la variable sobrepase el final de la tabla, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado (Nulo). • Indice de la variable que generó el error. • Error de Estado (Se especifica que se ha alcanzado en fin de la lista de valores.). **c)** En caso de que no se cuenten con los permisos de lectura, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado (Nulo). • Indice de la variable que generó el error. • Error de Estado (Se especifica que el Objeto Administrado no cuenta con permisos de lectura.).

A.3.4 Caso de Uso del Servicio SET-REQUEST

Caso de uso:	**Modificar información de un Objeto Administrado**
Actores:	Administrador SNMP (Iniciador), Agente SNMP (Sistema).
Propósito:	Modificar el valor de un Objeto Administrado.
Resumen:	El Administrador desea modificar la información de un Objeto Administrado, por lo cual envía una solicitud al Agente, ésta es atendida, y en caso de que se cuenten con los permisos de escritura, se le retorna al Administrador la información que ha sido modificada, en caso contrario se le retorna un mensaje de error.
Tipo:	Primario y real.

Sección: *Principal*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea modificar información de un Objeto Administrado.		
2.	El Administrador genera un mensaje de tipo SET-REQUEST el cual contiene los siguientes parámetros: • Identificador del PDU. • Error de Estado. • Indice de la Variable que generó el error. • Nombre del Objeto Administrado. • Valor del Objeto Administrado.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo SNMP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, y la almacena en una lista de espera.
		5.	Una vez que el Agente puede procesar la solicitud, éste le envía los parámetros necesarios al Administrador de la MIB para que realice la búsqueda del Objeto Administrado. Los parámetros son los

			siguientes: • Identificador del PDU. • Nombre del Objeto Administrado.
		6.	El Administrador de la MIB realiza la búsqueda. **a)** En caso de que existan, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que al menos alguno no exista, se genera un mensaje de error.
		7.	El Administrador de la MIB le envía la notificación generada al Agente.
		8.	El Agente recibe la notificación y genera un mensaje de tipo GET-RESPONSE con los mismos parámetros que la notificación, pero le añade un identificador de solicitud.
		9.	El Agente envía el mensaje al Administrador utilizando el protocolo SNMP.
9.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto Administrado. **a)** En caso de que se cuente con los permisos de escritura, se verifica si el Objeto Administrado es una tabla. - Si el Objeto Administrado es una tabla y en la solicitud se especifica un índice (el cual no sobrepasa los límites de la tabla), se reemplaza el valor que se encuentra en dicha posición. Además se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado.

			• Indice de la variable que generó el error (Nulo.). • Error de Estado. - Si el Objeto Administrado es una tabla y el índice va mas allá de sus límites de la tabla, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado (Nulo). • Indice de la variable que generó el error. • Error de Estado (Se especifica que el índice solicitado está más allá de los límites de la tabla.). - Si el Objeto Administrado no es una tabla, se sobrescribe el valor del Objeto Administrado y se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado. • Indice de la variable que generó el error (Nulo.). • Error de Estado (Nulo.). **b)** En caso de que no se cuente con los permisos de escritura, se genera una notificación con los siguientes parámetros: • Nombre del Objeto Administrado. • Valor del Objeto Administrado (Nulo). • Indice de la variable que generó el error. • Error de Estado (Se especifica que el Objeto Administrado no cuenta con permisos de escritura.).
		2.	Después de que se modificaron atributos en Objetos Administrados, estos cambios se deben de reflejar en los Recursos según sea el caso (Véase la sección

			"*Modificación de atributos en los Recursos*".).
Sección: *Modificación de atributos en los Recursos* **Curso normal de los eventos**			
	Acción del actor		**Respuesta del sistema**
		1.	Se busca el Recurso que está relacionado con el Objeto Administrado mediante una lista que representa esta relación. **a)** En caso de que no exista tal relación no se realiza ninguna acción. **b)** En caso de que exista la relación, se le envía a la Administración Virtual de Hardware un reporte en el que se especifica que hubo un cambio de valor en un Recurso. El Reporte va a contener los siguientes parámetros: • Identificador del Recurso. • Identificador de la Acción. • Lista de Atributos a modificar.
		2.	La Administración Virtual de Hardware realiza una búsqueda del Recurso. **a)** En caso de que no exista, se genera un reporte de falla al Administrador de la MIB (Véase el Caso de Uso "*Reportar alarmas generadas en los Recursos*".). **b)** En caso de que exista se cambian los valores del Recurso que correspondan.

A.4 Casos Abstractos de Uso del Modelo OSI

Los Casos Abstractos de Uso del Modelo OSI son el resultado de realizar la "A*bstracción Vertical*" en este modelo de información. Estos casos abstractos de uso son dos, los cuales representan la fusión entre los casos de uso reales del modelo. El primero de ellos es el caso "*Ejecutar un servicio en el Agente (Servicio CMIS)*", el cual representa la fusión entre los siguientes casos de uso:

- Obtener información de Objetos Administrados. (*Servicio M-GET*)
- Cancelar una solicitud de información. (*Servicio M-CANCEL-GET*)
- Modificar información de Objetos Administrados. (*Servicio M-SET*)
- Invocar servicios en un Objeto Administrado. (*Servicio M-ACTION*)
- Crear un Objeto Administrado. (*Servicio M-CREATE*)
- Borrar un Objeto Administrado. (*Servicio M-DELETE*)

El caso de uso restante "*Reportar eventos generados en los Recursos (Modelo OSI)*", no obstante que no forma parte de algún grupo de casos de uso, se debe abstraer de cualquier forma, esto con el fin de omitir detalles innecesarios para la aplicación de la "*Abstracción Horizontal*", tales como los parámetros que se envían para realizar una acción determinada.

La descripción de los casos abstractos de uso del Modelo OSI se presentan a continuación.

A.4.1 Caso Abstracto de Uso del Servicio CMIS

Caso de uso:	**Ejecutar un servicio en el Agente.**
Actores:	Administrador CMIP (Iniciador), Agente CMIP (Sistema).
Propósito:	Ejecutar acciones sobre Objetos Administrados.
Resumen:	El Administrador desea ejecutar un servicio determinado sobre uno o varios Objetos Administrados. El Administrador envía información referente al servicio solicitado al Agente, éste lo interpreta, ejecuta y retorna los resultados al Administrador
Tipo:	Primario y esencial.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	El caso de uso comienza cuando el Administrador desea ejecutar un servicio en el Agente. Los servicios pueden ser: • M-GET. • M-CANCEL-GET. • M-SET. • M-ACTION. • M-CREATE. • M-DELETE.		
2.	El Administrador genera un mensaje, el cual representa el servicio que se debe atender.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo CMIP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud. Si así lo requiere, almacena la solicitud en una lista de espera y guarda también el Identificador de Invocación en otra lista. En caso contrario la atiende inmediatamente (Sólo servicio M-CANCEL-GET).
		5.	Una vez que el Agente puede procesar la solicitud, y en caso de que se requiera realizar una acción sobre Objetos Administrados, el Agente envía varios parámetros al Administrador de la MIB

			para que realice la búsqueda de los Objetos Administrados sobre los cuales se debe ejecutar la acción. En caso contrario se trata de una cancelación de una solicitud de información, por lo cual se cancela el proceso que esté atendiendo dicha solicitud.
		6.	El Administrador de la MIB realiza la búsqueda de los Objetos Administrados. **a)** En caso de que existan, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que al menos uno no exista, se genera una notificación para cada Objeto no existente, con los errores correspondientes.
		7.	El Administrador de la MIB envía las notificaciones generadas al Agente.
		8.	El Agente recibe las notificaciones y genera el mensaje pertinente.
		9.	El Agente envía al Administrador cada uno de los mensajes, utilizando el protocolo CMIP y elimina el identificador de invocación de la lista correspondiente.
10.	El Administrador recibe los mensajes, e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró todos los Objetos Administrados.
		2.	Para cada Objeto Administrado, se analiza que éste exista y que tenga permisos correspondientes para ejecutar la acción. **a)** En caso de que se cuente con los permisos, se ejecuta la acción y se genera una notificación, la cual informa del éxito o no de la acción. **b)** En caso contrario, se genera una notificación la cual informa que no se

			cuentan con los permisos necesarios para ejecutar la acción.

A.4.2 Caso Abstracto de Uso del Servicio M-EVENT-REPORT

Caso de uso:	**Reportar eventos generados en los Recursos**
Actores:	Recurso (Iniciador), Agente CMIP (Sistema), Administrador CMIP.
Propósito:	Reportar al Administrador los cambios de estado que han experimentado los Recursos actualmente.
Resumen:	Un Recurso experimenta un cambio de estado. El Objeto Administrado que representa al Recurso es notificado de esta situación a través del Administrador Virtual de Hardware, genera la notificación correspondiente y se la hace llegar tanto a la Bitácora como al Discriminador de Envío de Eventos. Si la notificación cumple con los criterios de filtrado, ésta es convertida en un mensaje y es enviada al Administrador. En caso contrario, es desechada.
Tipo:	Primario y esencial.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando un Recurso experimenta una falla.		
		2.	La Administración Virtual de Hardware detecta la falla y envía un reporte al Administrador de la MIB.
		3.	El Administrador de la MIB relaciona el Identificador del Recurso el Objeto Administrado correspondiente, y le envía a cada uno de ellos el reporte generado.
		4.	El Objeto Administrado analiza el reporte y determina la severidad de la alarma.
		5.	El Objeto Administrado genera una notificación.
		6.	El Objeto Administrado envía la notificación a las siguientes entidades: • Bitácora (Consúltese la sección "*Bitácora*"). • Discriminador de Envío de Eventos (Consúltese la sección "*Discriminador*

			de Envío de Eventos").
		7.	Si la notificación no es desechada por el Discriminador, el Agente la recibe y genera un mensaje.
		8.	El Agente envía el mensaje al Administrador utilizando el protocolo CMIP y en caso de que el parámetro "Modo" haya sido de tipo "Confirmado" (si así es requerido), almacena en una lista el Identificador de Invocación del mensaje.
9.	El Administrador recibe el mensaje, lo analiza y le da el seguimiento correspondiente. **a)** En caso de que el parámetro "Modo" sea de tipo "*Confirmado*", consúltese la sección "*Respuesta al Evento*". **b)** En caso de que el parámetro "Modo" sea de tipo "No Confirmado", no se genera ningún mensaje.		
Sección: *Bitácora* **Curso normal de los eventos:**			
	Acción del actor		**Respuesta del sistema**
		1.	El Filtro de Procesamiento de Bitácora determina si la notificación debe ser almacenada en la Bitácora, o debe ser desechada. **a)** Si la notificación no es desechada, es almacenada en la Bitácora como un Objeto Administrado. **b)** En caso contrario, la notificación es desechada.
Sección: *Discriminador de Envío de Eventos* **Curso normal de los eventos:**			
	Acción del actor		**Respuesta del sistema**
		1.	El Discriminador de Envío de Eventos determina si la notificación debe enviarse al Administrador, o debe ser desechada.

			a) Si la notificación no es desechada, se envía al Agente. **b)** En caso contrario, la notificación es desechada.
Sección: *Respuesta al Evento*			
Curso normal de los eventos:			
	Acción del actor		**Respuesta del sistema**
1.	El Administrador genera un mensaje.		
2.	El Administrador envía el mensaje al Agente mediante el protocolo CMIP.		
		3.	El Agente recibe el mensaje, determina el tipo de solicitud, y realiza una búsqueda del Identificador de Invocación en su lista de Identificadores para relacionarlo con un evento generado con anterioridad. **a)** En caso de que el identificador exista en la lista, se verifica el parámetro "Respuesta al Evento", y dependiendo de lo preestablecido en la configuración del Agente, se realiza la acción que corresponda. **b)** En caso contrario el mensaje es desechado.

A.5 Casos Abstractos de Uso del Modelo de Internet

A continuación se presenta el resultado de la Abstracción Vertical de los Casos de Uso del Modelo de Internet, el cual consiste de dos casos abstractos de uso. El primero de ellos es el caso "*Ejecutar un Servicio en el Agente (Modelo de Internet)*", el cual representa la fusión entre los siguientes casos de uso:

- Obtener el valor de un Objeto Administrado. (*Servicio GET-REQUEST*)
- Obtener el valor siguiente de un Objeto Administrado. *(Servicio GET-NEXT-REQUEST)*
- Modificar información de un Objeto Administrado. (*Servicio SET-REQUEST*)

El caso de uso restante ("*Reportar eventos generados en los Recursos (Modelo de Internet)*"), no obstante que no forma parte de algún grupo de casos de uso, se debe abstraer de cualquier forma, con el fin de omitir detalles innecesarios para la aplicación de la "*Abstracción Horizontal*", tales como los parámetros que se envían para realizar una acción determinada.

La descripción de los casos abstractos de uso del Modelo de Internet se presentan a continuación.

A.5.1 Caso Abstracto de Uso del Servicio SNMP

Caso de uso:	**Ejecutar un Servicio en el Agente**
Actores:	Administrador SNMP (Iniciador), Agente SNMP (Sistema).
Propósito:	Obtener el valor actual de un Objeto Administrado.
Resumen:	El Administrador desea ejecutar un servicio determinada sobre un Objeto Administrado.
Tipo:	Primario y esencial.

Sección: *Principal*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea ejecutar un servicio sobre un Objeto Administrado. Los servicios pueden ser: • GET-REQUEST. • GET-NEXT-REQUEST. • SET-REQUEST.		
2.	El Administrador genera un mensaje, el cual representa el servicio que se debe despachar.		
3.	El Administrador envía el mensaje al Agente mediante el protocolo SNMP.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud, y la almacena en una lista de espera.
		5.	Una vez que el Agente puede procesar la solicitud, éste le envía los parámetros necesarios al Administrador de la MIB para la búsqueda del Objeto Administrado.
		6.	El Administrador de la MIB realiza la búsqueda del Objeto Administrado. **a)** En caso de que exista, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que no exista, se genera un mensaje de error.
		7.	El Administrador de la MIB le envía la

			notificación generada al Agente.
		8.	El Agente recibe la notificación y genera un mensaje de tipo GET-RESPONSE con los mismos parámetros que la notificación, pero le añade un identificador de solicitud.
		9.	El Agente envía el mensaje al Administrador utilizando el protocolo SNMP.
10.	El Administrador recibe el mensaje e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró el Objeto Administrado. **a)** En caso de que se cuenten con los permisos necesarios, se realiza la acción requerida por el Administrador. Se genera una notificación, la cual contiene el valor del Objeto Administrado sobre el cual se realizó la acción. **b)** En caso contrario, se genera una notificación, la cual especifica el porqué no se pudo realizar la acción.

A.5.2 Caso Abstracto de Uso del Servicio TRAP

Caso de uso:	**Reportar eventos generados en los Recursos**
Actores:	Recurso(iniciador), Agente SNMP(Sistema), Administrador SNMP.
Propósito:	Reportar al Administrador los cambios de estado que han experimentado los Recursos actualmente.
Resumen:	Un Recurso experimenta un cambio de estado. La Administración Virtual reporta esta falla al Administrador de la MIB, este relaciona al Recurso con un Objetos Administrado y modifica su valor de acuerdo al reporte. El objeto modificado le indica al Agente esta situación. El Agente verifica los valores frontera de los objetos y si estos rebasan alguno de ellos, se genera un TRAP el cual es enviado al Administrador.
Tipo:	Primario y real.

Sección: *Principal*

Curso normal de los eventos

	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando un Recurso experimenta una cambio de estado.		
		2.	La Administración Virtual de Hardware detecta la falla y envía un reporte al Administrador de la MIB.
		3.	El Administrador de la MIB relaciona al Recurso con un Objeto Administrado y modifica su valor.
		4.	Al modificarse el valor de un Objeto Administrado, este le envía una notificación al Agente de este suceso.
		5.	El Agente analiza este cambio de valor utilizando un esquema de filtrado. **a)** En caso de que el valor rebase los límites frontera se genera un mensaje de tipo TRAP. **b)** En caso de que el valor no rebase los límites frontera, no se realiza ninguna acción, finalizando el caso de uso.
7.	El Administrador recibe el mensaje y le da el seguimiento pertinente.	**6**.	El Agente envía el mensaje utilizando el protocolo SNMP.

A.6 Casos de Uso del Dominio

A continuación se presenta el resultado de la Abstracción Horizontal de los Casos de Uso, el cual consiste de los Casos de Uso del Dominio, los cuales son:

- Ejecutar un Servicio en el Agente.
- Reportar eventos generados en los Recursos.

El Caso de Uso del Dominio "*Ejecutar un Servicio en el Agente*" representa la fusión de los siguientes casos abstractos de uso:

- Ejecutar un Servicio en el Agente (Servicio CMIS).
- Ejecutar un Servicio en el Agente (Modelo de Internet).

El Caso de Uso del Dominio "*Reportar eventos generados por los Recursos*" representa la fusión de los siguientes casos abstractos de uso:

- Reportar eventos generados por los Recursos (Modelo OSI).
- Reportar eventos generados por los Recursos (Modelo de Internet).

La descripción de los casos de uso del dominio se presenta a continuación:

A.6.1 Caso de Uso del Dominio "Ejecutar un Servicio en el Agente"

Caso de uso:	**Ejecutar un Servicio en el Agente**
Actores:	Administrador(iniciador), Agente(Sistema).
Propósito:	Ejecutar acciones sobre Objetos Administrados.
Resumen:	El Administrador desea ejecutar un servicio sobre uno o varios Objetos Administrados, por lo cual envía una solicitud la cual es analizada por el Agente, ejecuta el servicio correspondiente y retorna los resultados de la ejecución al Administrador.
Tipo:	Primario y esencial.

Sección: *Principal*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando el Administrador desea ejecutar un servicio sobre uno o varios Objetos Administrados.		
2.	El Administrador genera un mensaje, el cual representa la acción que se debe realizar.		
3.	El Administrador envía el mensaje al Agente mediante un protocolo de Administración de Red.		
		4.	El Agente recibe el mensaje, determina el tipo de solicitud. Si así lo requiere, almacena la solicitud en una lista de espera y guarda también el Identificador de Invocación en otra lista. En caso contrario la atiende inmediatamente.
		5.	Una vez que el Agente puede procesar la solicitud, y en caso de que se requiera realizar una acción sobre Objetos Administrados, el Agente envía varios parámetros al Administrador de la MIB para que realice la búsqueda de los Objetos Administrados sobre los cuales se debe ejecutar la acción. En caso contrario se trata de una cancelación de una solicitud de información, por lo cual se realiza una búsqueda del proceso que

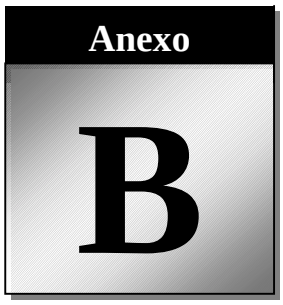
			procesa dicha solicitud, y se cancela.
		6.	El Administrador de la MIB realiza la búsqueda de los Objetos Administrados. **a)** En caso de que existan, consúltese la sección *Búsqueda Exitosa*. **b)** En caso de que al menos uno no exista, se genera una notificación para cada Objeto no existente, con los errores correspondientes.
		7.	El Administrador de la MIB envía las notificaciones generadas al Agente.
		8.	El Agente recibe las notificaciones y genera el mensaje pertinente.
		9.	El Agente envía al Administrador cada uno de los mensajes, utilizando un protocolo de administración de red y elimina el identificador de invocación de la lista correspondiente.
10.	El Administrador recibe los mensajes, e interpreta su información.		
Sección: *Búsqueda Exitosa*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
		1.	El Administrador de la MIB encontró todos los Objetos Administrados.
		2.	Para cada Objeto Administrado, se analiza que éste exista y que tenga permisos correspondientes para ejecutar la acción. **a)** En caso de que se cuente con los permisos necesarios, se ejecuta la acción y se genera una notificación, la cual informa del éxito o no de la acción. **b)** En caso contrario, se genera una notificación la cual informa que no se cuentan con los permisos necesarios para ejecutar la acción.

A.6.2 Caso de Uso del Dominio "Reportar Eventos Generados por Recursos"

Caso de uso:	**Reportar eventos generados por los Recursos**
Actores:	Recurso(iniciador), Agente(Sistema), Administrador.
Propósito:	Reportar al Administrador los cambios de estado que han experimentado los Recursos actualmente.
Resumen:	Un Recurso experimenta un cambio de estado. El Objeto Administrado que representa al Recurso es notificado de esta situación a través del Administrador Virtual de Hardware, genera la notificación correspondiente, ésta es convertida en un mensaje y es enviada al Administrador por parte del Agente. En caso contrario, es desechada.
Tipo:	Primario y esencial.

Sección: *Principal*			
Curso normal de los eventos			
	Acción del actor		**Respuesta del sistema**
1.	Este caso de uso comienza cuando un Recurso experimenta un cambio de estado.		
		2.	La Administración Virtual de Hardware detecta el evento y envía un reporte al Administrador de la MIB.
		3.	El Administrador de la MIB relaciona el Identificador del Recurso con el Objeto Administrado correspondiente, y en su caso se modifican el o los valores que correspondan en el Objeto Administrado.
		4.	Los Objetos Administrados reportan éste cambio de valores a través de una notificación. Esta notificación es enviada a un proceso de la MIB.
		5.	El proceso de la MIB decide enviar directamente al Agente esta notificación o si es necesario enviarla hacia uno o más procesos adicionales que permitan analizar la notificación.
		6.	Si la notificación no es desechada, el

		7.	Agente la recibe y genera un mensaje, el cual contiene información sobre el cambio de estado del Recurso. El Agente envía el mensaje al Administrador utilizando un protocolo de administración de red.
8.	El Administrador recibe el mensaje, lo analiza y le da el seguimiento correspondiente.		



Diagramas Conceptuales de Colaboración del Dominio

B.1 Introducción

En el presente anexo se presentan los Diagramas Conceptuales de Colaboración basados en los Casos de Uso de los modelos de información OSI e Internet. En ellos se presentan las acciones que son ejecutadas por los conceptos que intervienen en un determinado caso de uso.

Por último se presentan los Diagramas Conceptuales Abstractos, los cuales son el resultado de aplicar la "*Abstracción Vertical*" en cada modelo de información.

B.2 Diagramas Conceptuales de Colaboración del Modelo OSI

Los Diagramas Conceptuales de Colaboración del Modelo OSI corresponden a los casos de uso de este modelo de información (Véase el "*Anexo A*" en su sección "*A.2 Casos de Uso del Modelo OSI*"), en donde se presentan los conceptos que intervienen en cada caso de uso además de las acciones que son llevadas a cabo entre ellos.

A continuación se presentan éstos diagramas.

B.2.1 Servicio M-EVENT-REPORT

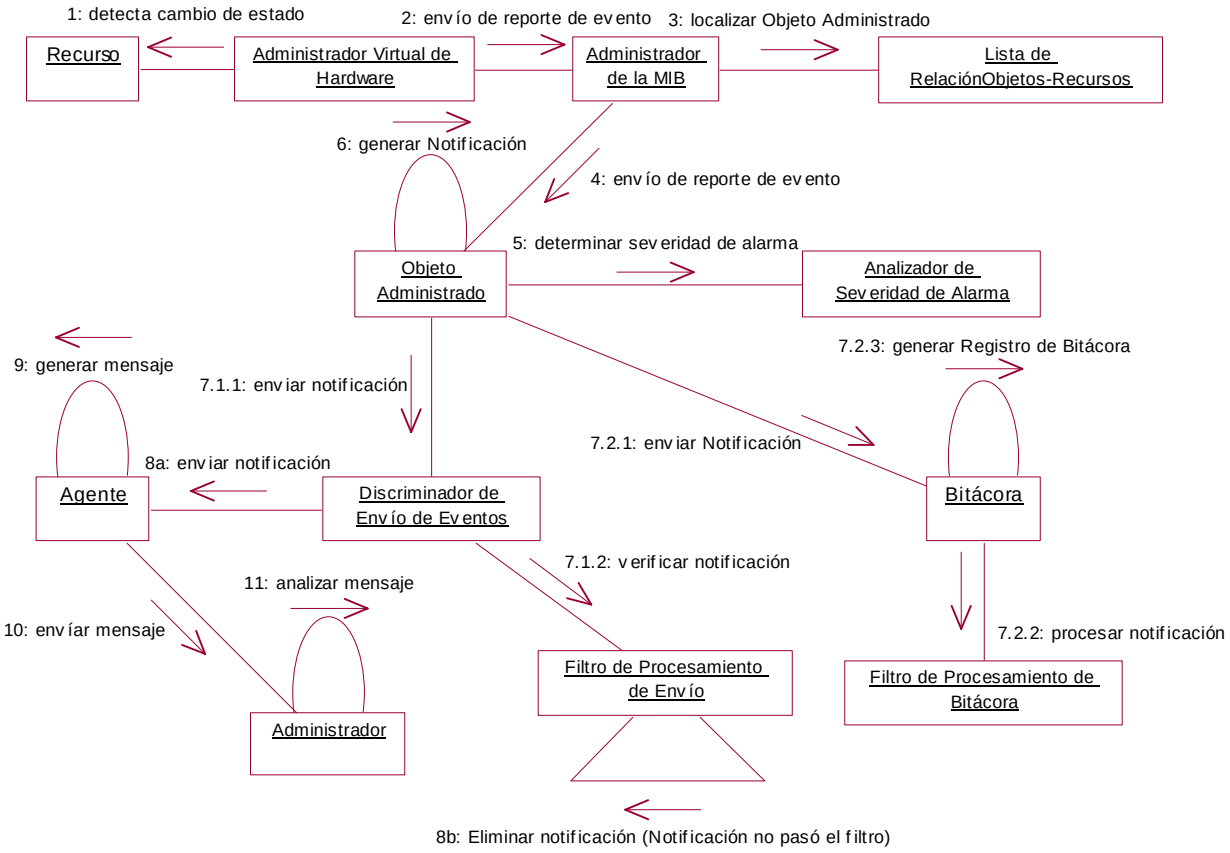


Figura B.1 Diagrama Conceptual de Colaboración del Servicio "M-EVENT-REPORT".

B.2.2 Servicio M-GET

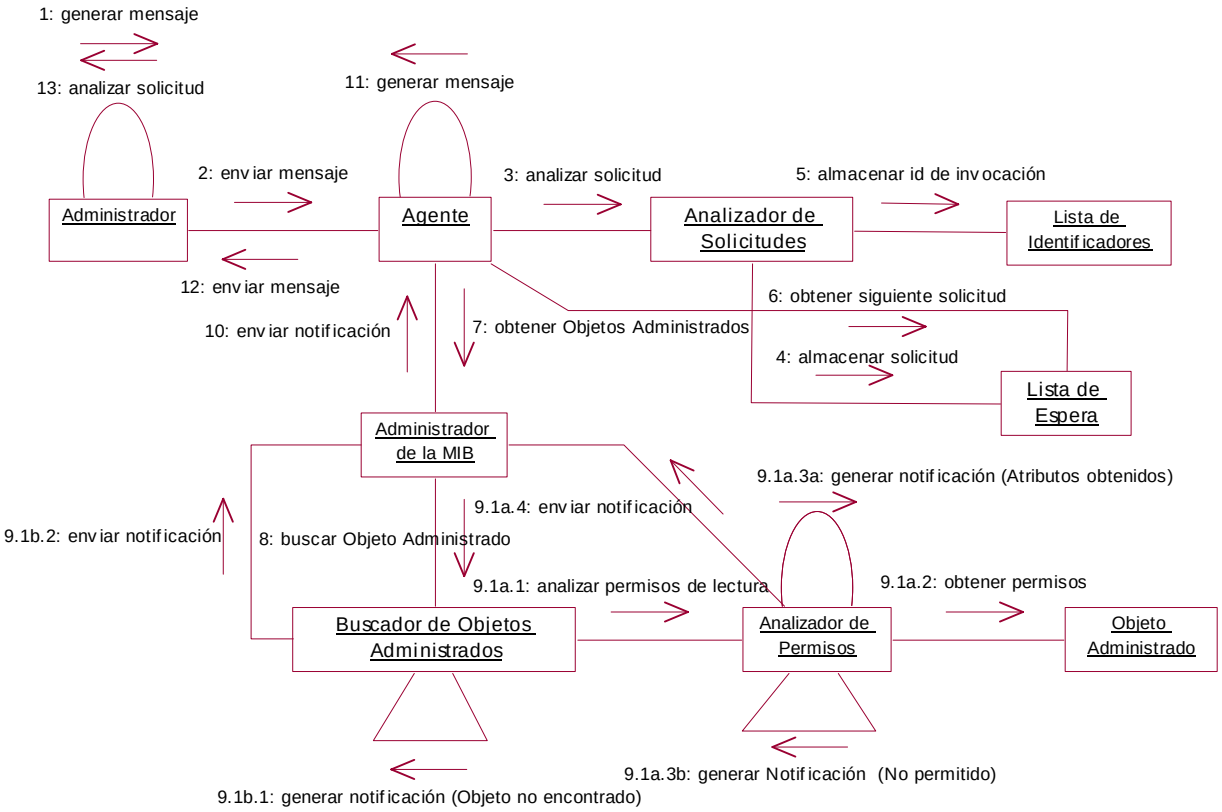


Figura B.2 Diagrama Conceptual de Colaboración del Servicio "M-GET".

B.2.3 Servicio M-CANCEL-GET

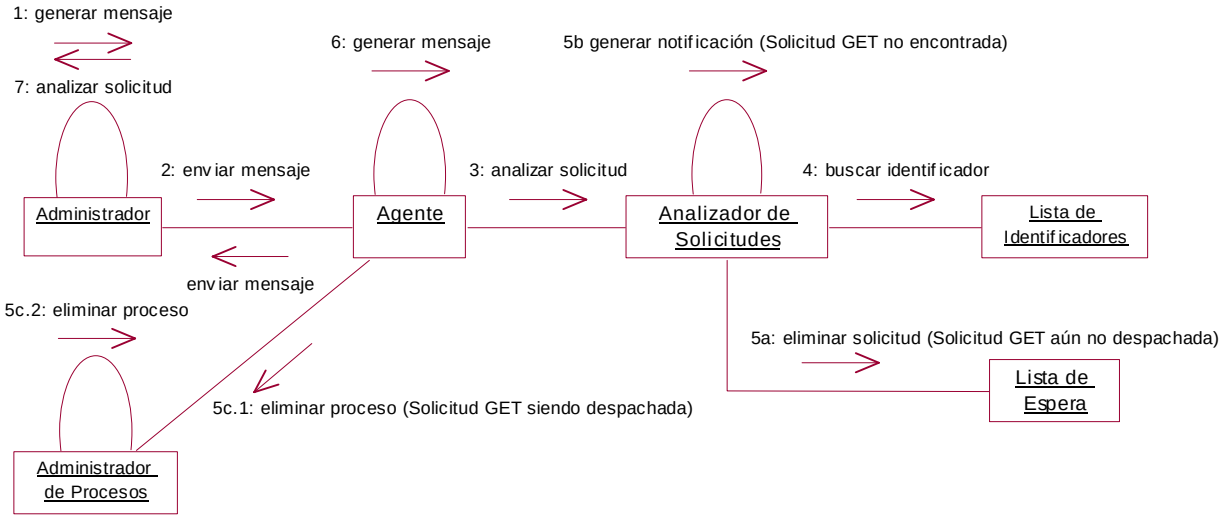


Figura B.3 Diagrama Conceptual de Colaboración del Servicio "M-CANCEL-GET".

B.2.4 Servicio M-SET

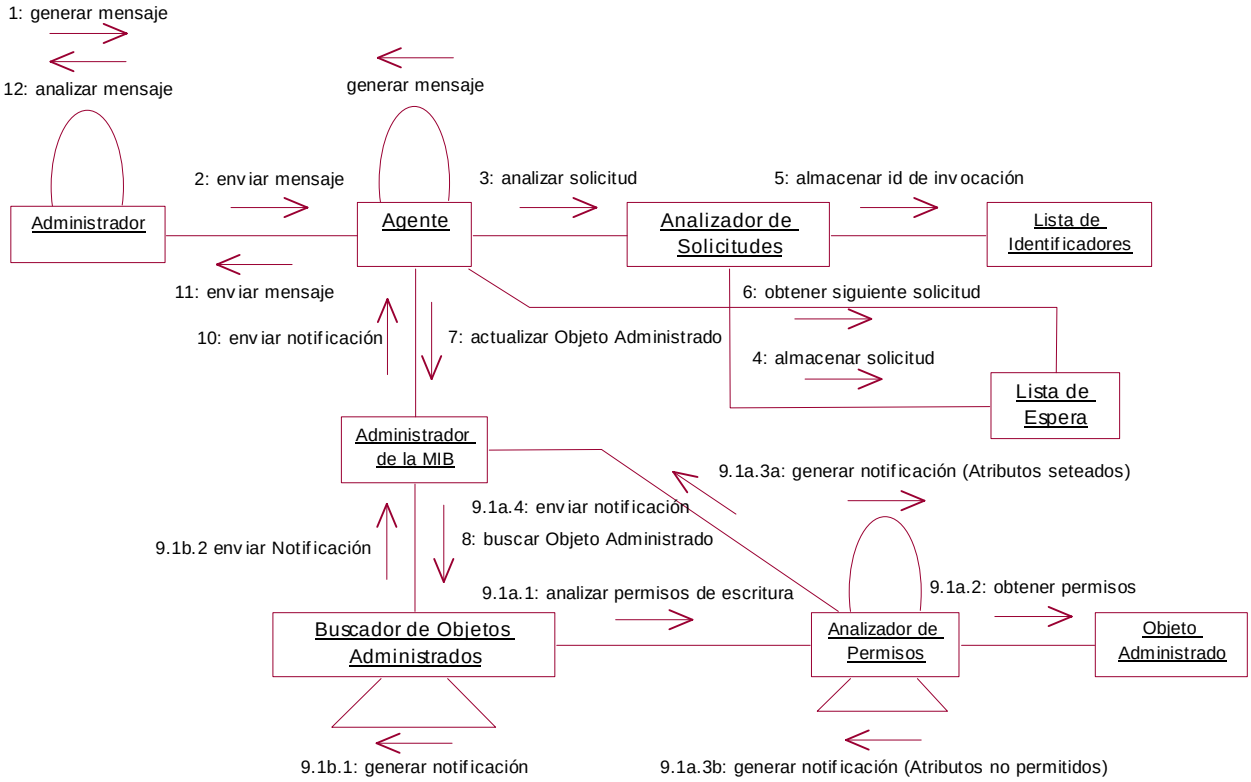


Figura B.4 Diagrama Conceptual de Colaboración del Servicio "M-SET".

B.2.5 Servicio M-ACTION

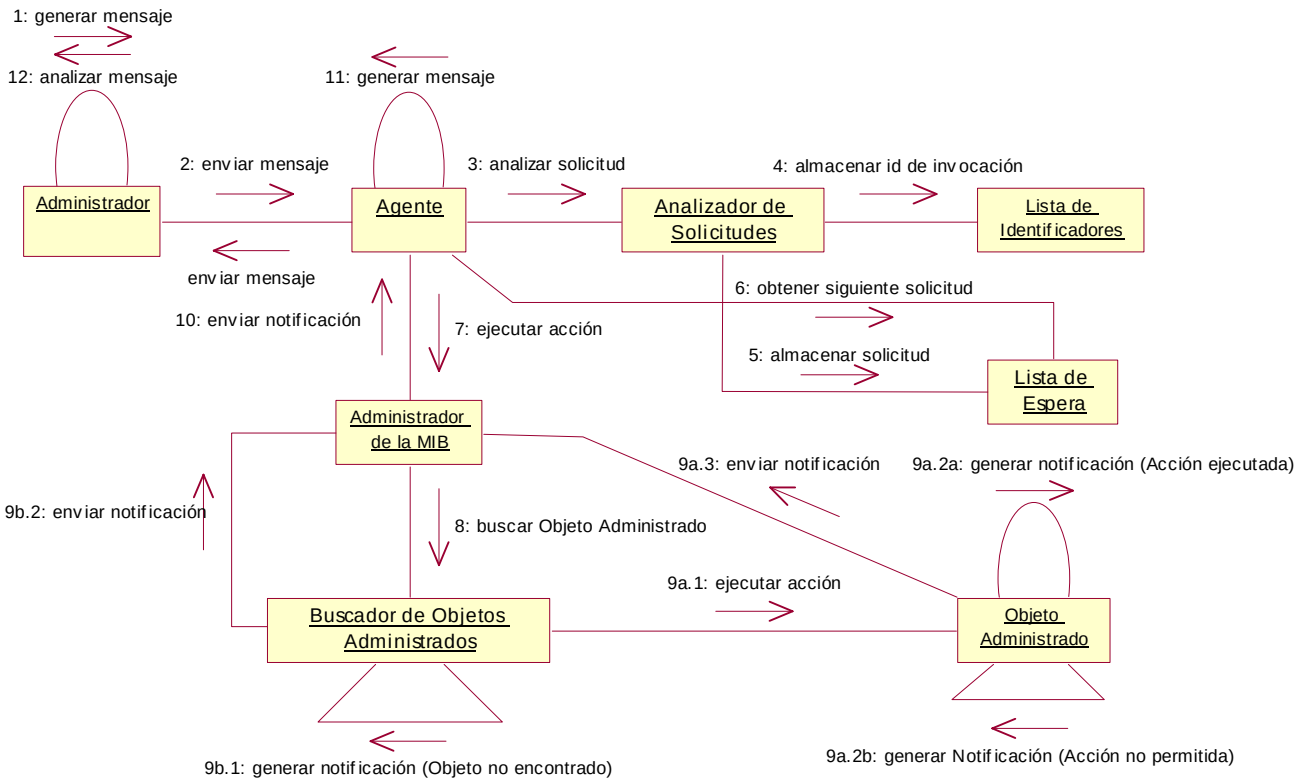


Figura B.5 Diagrama Conceptual de Colaboración del Servicio "M-ACTION".

B.2.6 Servicio M-CREATE

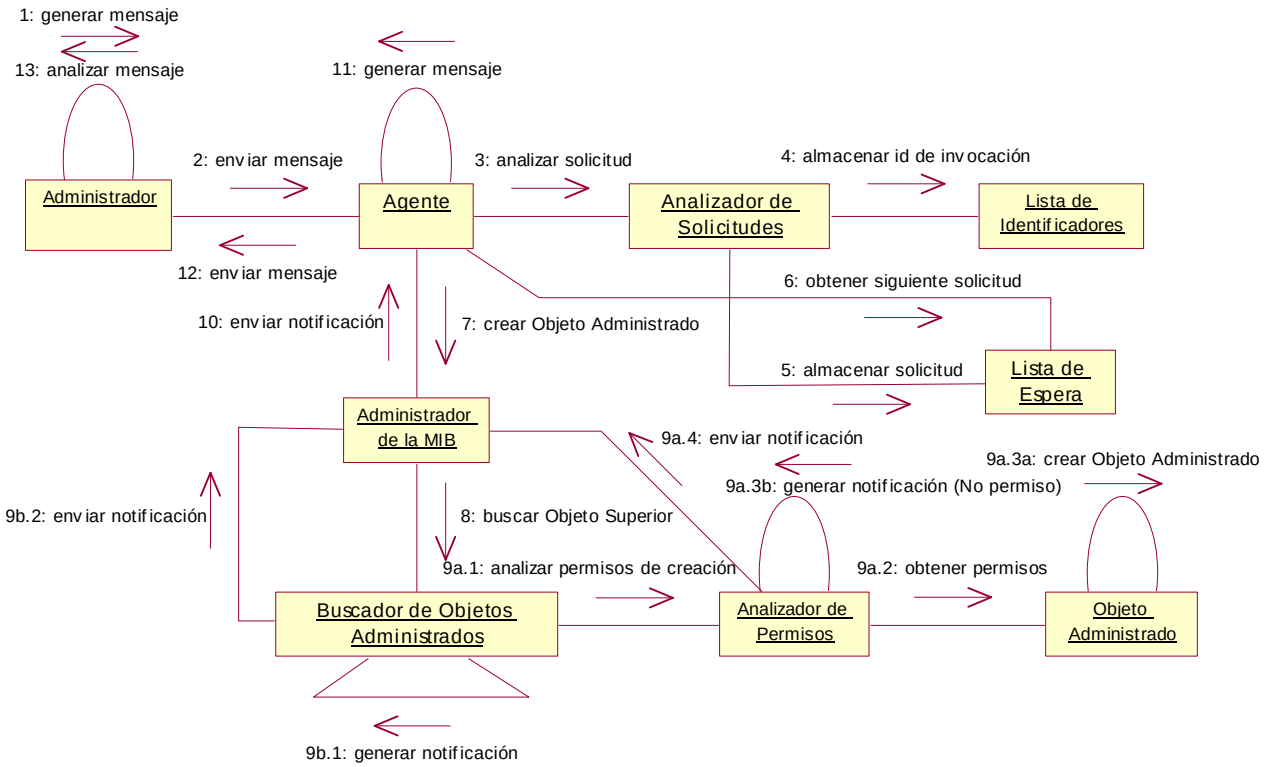


Figura B.6 Diagrama Conceptual de Colaboración del Servicio "M-CREATE".

B.2.7 Servicio M-DELETE

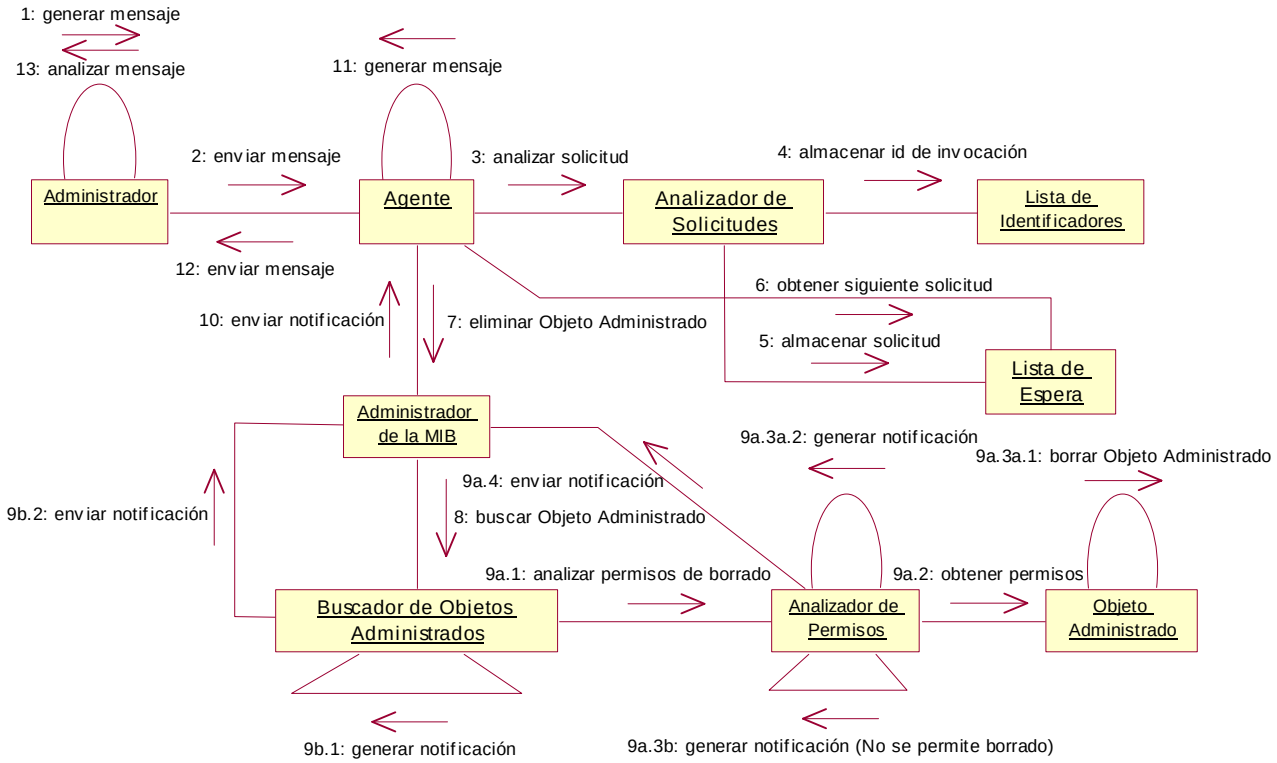


Figura B.7 Diagrama Conceptual de Colaboración del Servicio "M-DELETE".

B.3 Diagramas Conceptuales de Colaboración del Modelo de Internet

Los Diagramas Conceptuales de Colaboración del Modelo de Internet corresponden a los casos de uso de este modelo de información (Véase el "*Anexo A*" en su sección "*A.3 Casos de Uso del Modelo de Internet*"), en donde se presentan los conceptos que intervienen en cada caso de uso además de las acciones que son llevadas a cabo entre ellos.

A continuación se presentan éstos diagramas.

B.3.1 Servicio TRAP

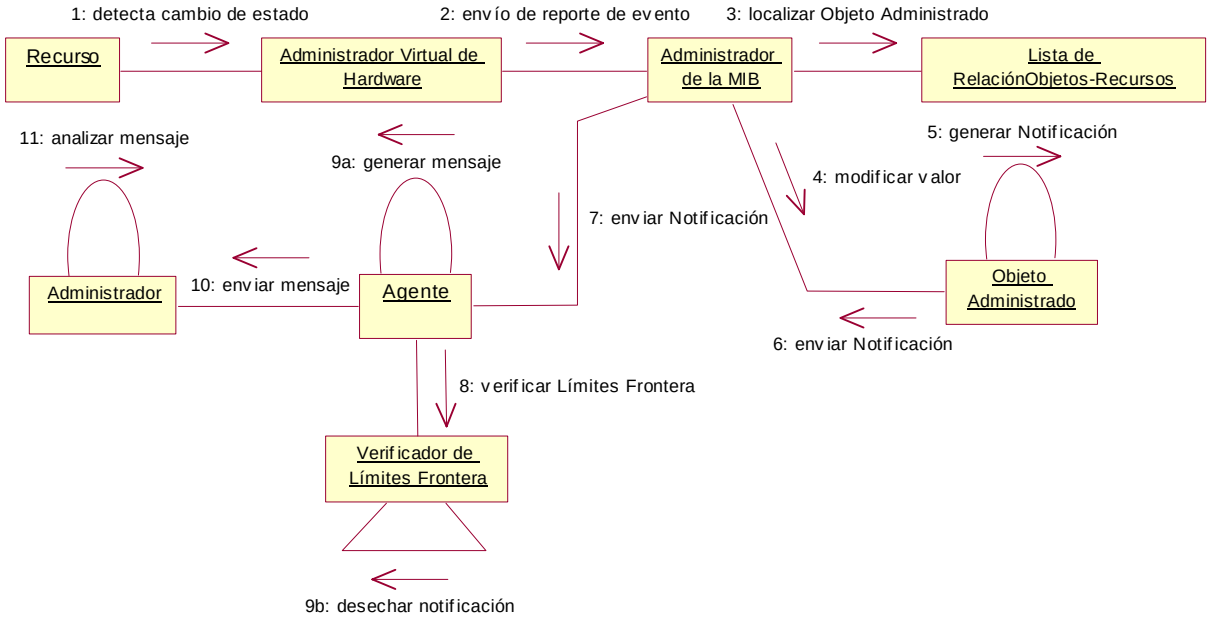


Figura B.8 Diagrama Conceptual de Colaboración del Servicio "TRAP".

B.3.2 Servicio GET-REQUEST

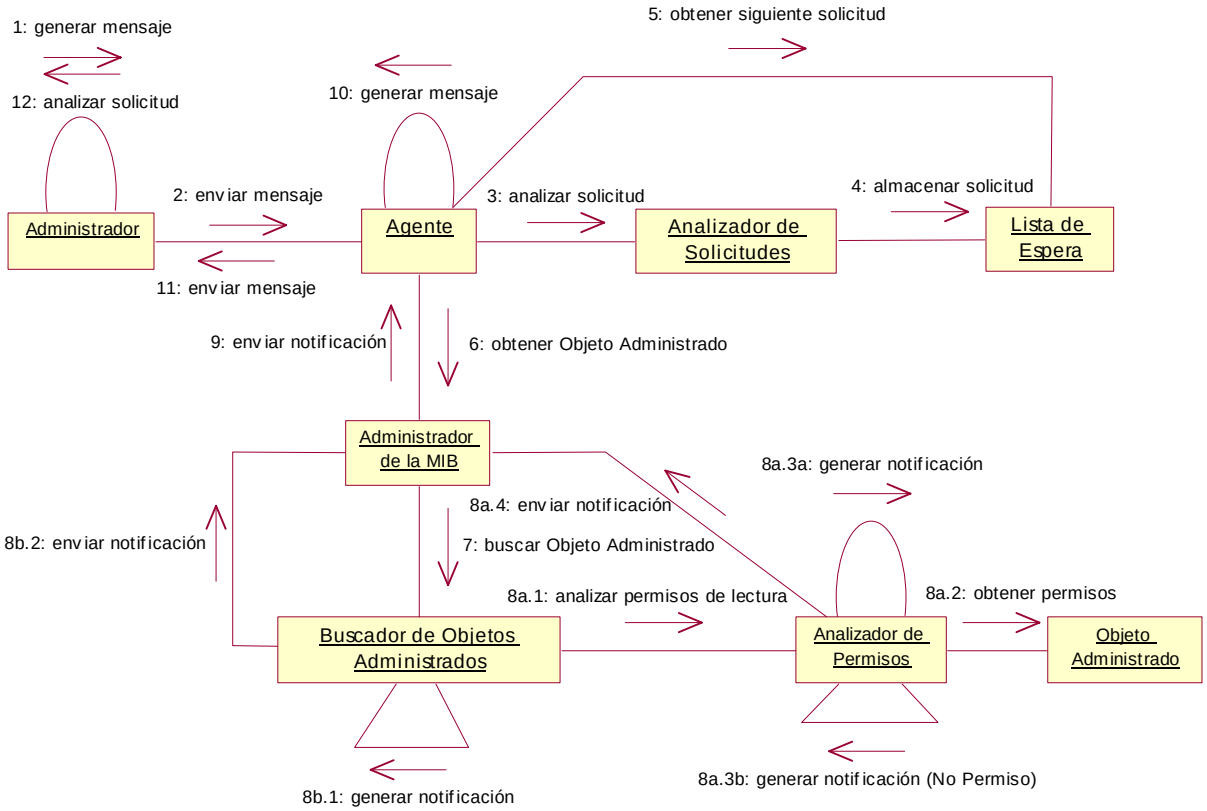


Figura B.9 Diagrama Conceptual de Colaboración del Servicio "GET-REQUEST".

B.3.3 Servicio GET-NEXT-REQUEST

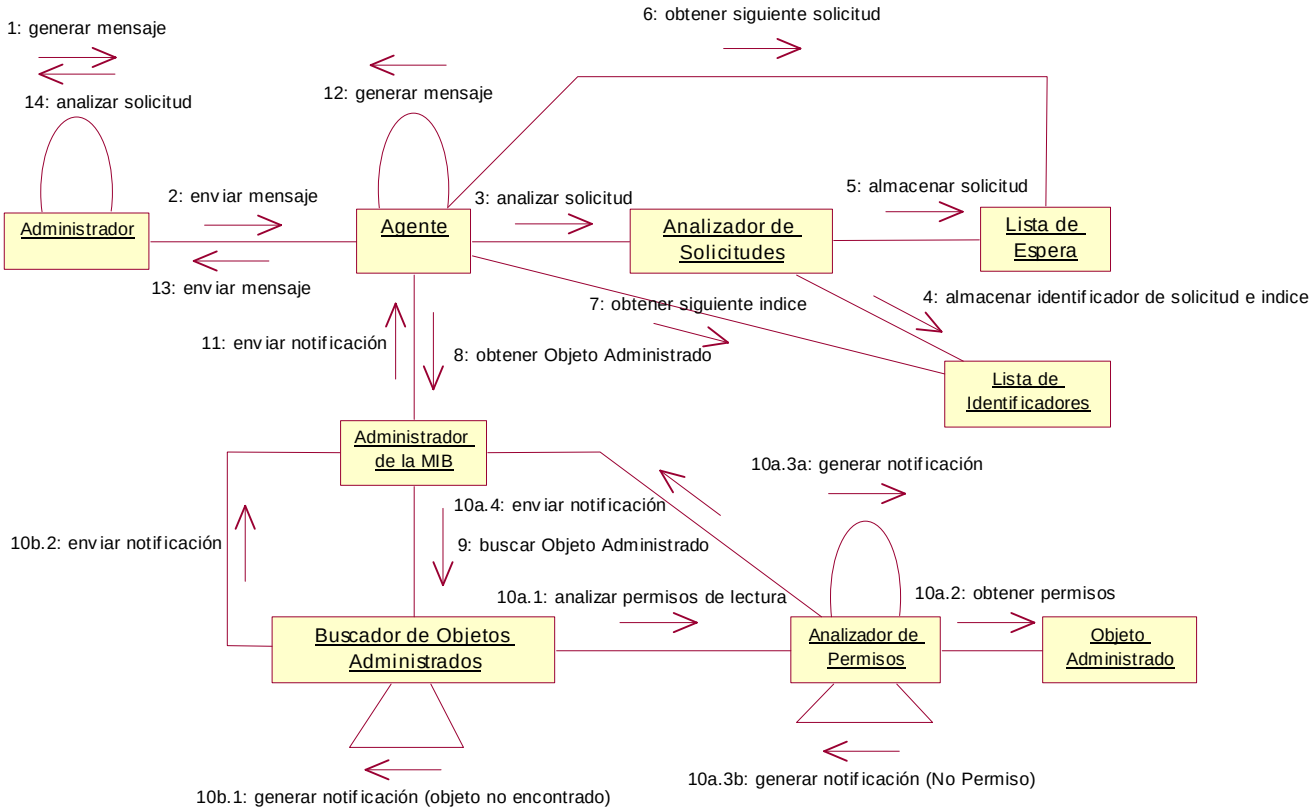


Figura B.10 Diagrama Conceptual de Colaboración del Servicio "GET-NEXT-REQUEST".

B.3.4 Servicio SET-REQUEST

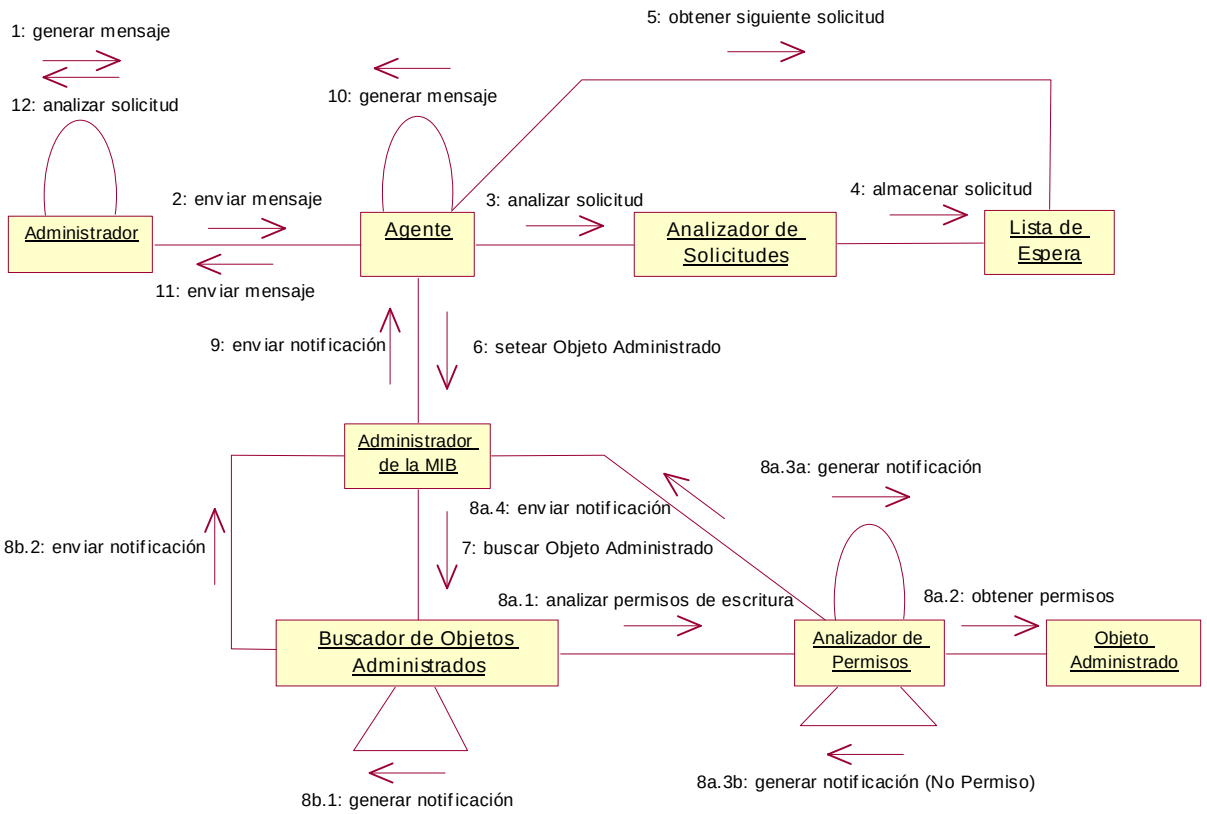


Figura B.11 Diagrama Conceptual de Colaboración del Servicio "SET-REQUEST".

B.4 Diagramas Conceptuales Abstractos del Modelo OSI

Los Diagramas Conceptuales Abstractos del Modelo OSI corresponden a los casos abstractos de uso de este modelo de información (Véase el "*Anexo A*" en su sección "*A.4 Casos Abstractos de Uso del Modelo OSI*"), los cuales son el resultado de aplicar la "*Abstracción Horizontal*".

A continuación se presentan éstos diagramas.

B.4.1 Servicio CMIS

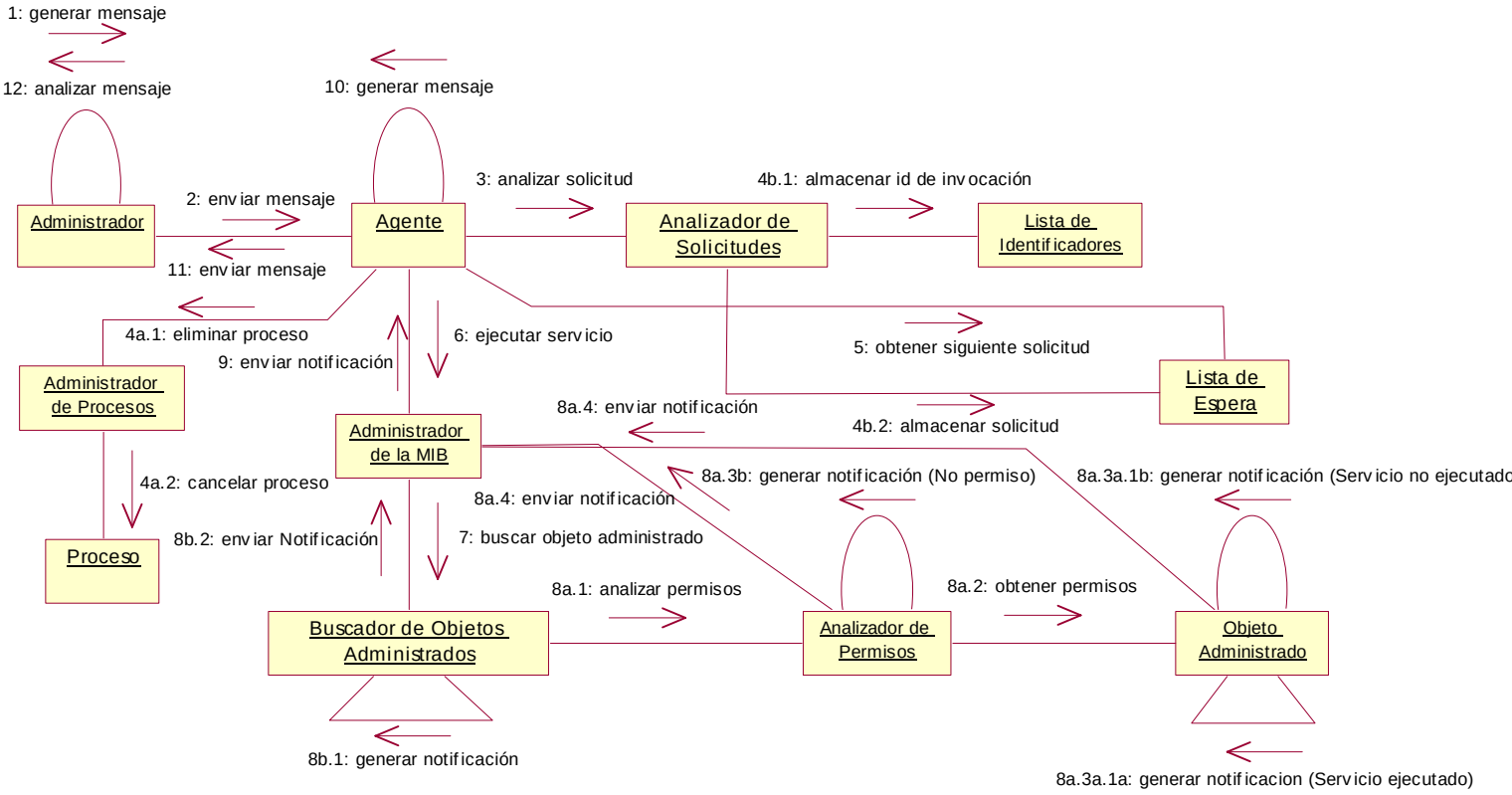


Figura B.12 Diagrama Conceptual Abstracto del Servicio CMIS.

B.4.2 Reporte de Evento (MODELO OSI)

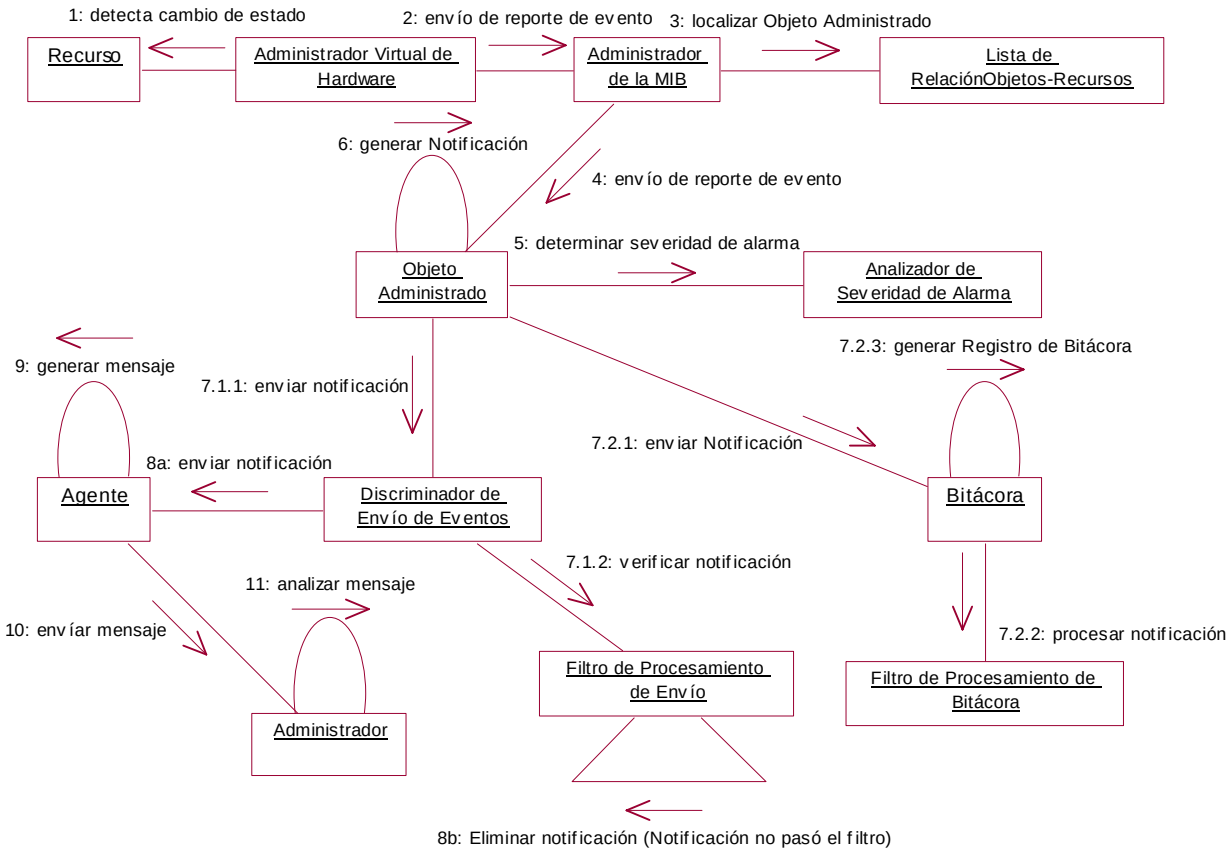


Figura B.13 Diagrama Conceptual Abstracto del Reporte de Evento (Modelo OSI).

B.5 Diagramas Conceptuales Abstractos del Modelo de Internet

Los Diagramas Conceptuales Abstractos del Modelo de Internet corresponden a los casos abstractos de uso de este modelo de información (Véase el "*Anexo A*" en su sección "*A.5 Casos Abstractos de Uso del Modelo de Internet*"), los cuales son el resultado de aplicar la "*Abstracción Horizontal*".

A continuación se presentan éstos diagramas.

B.5.1 Servicio SNMP

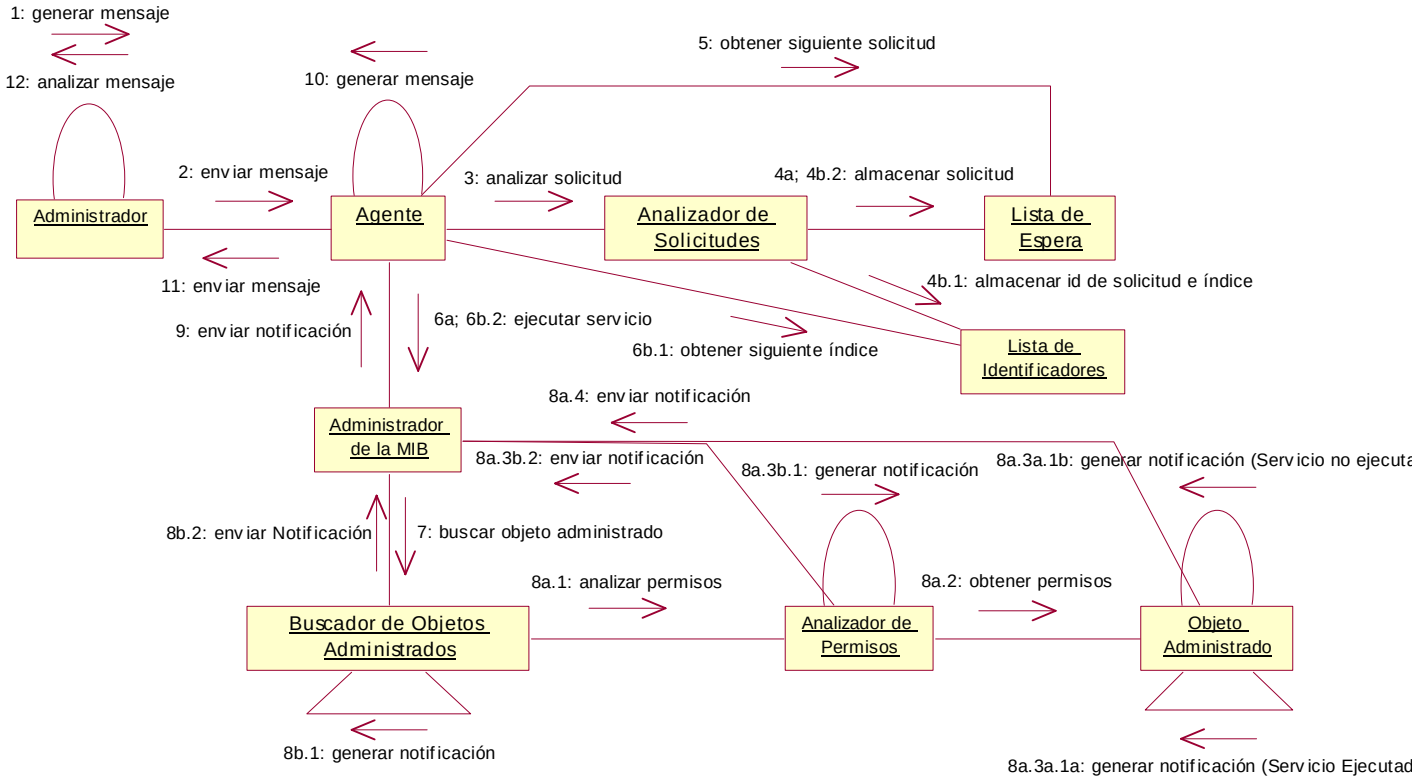


Figura B.14 Diagrama Conceptual Abstracto del Servicio SNMP.

B.5.2 Reporte de Evento (Modelo de Internet)

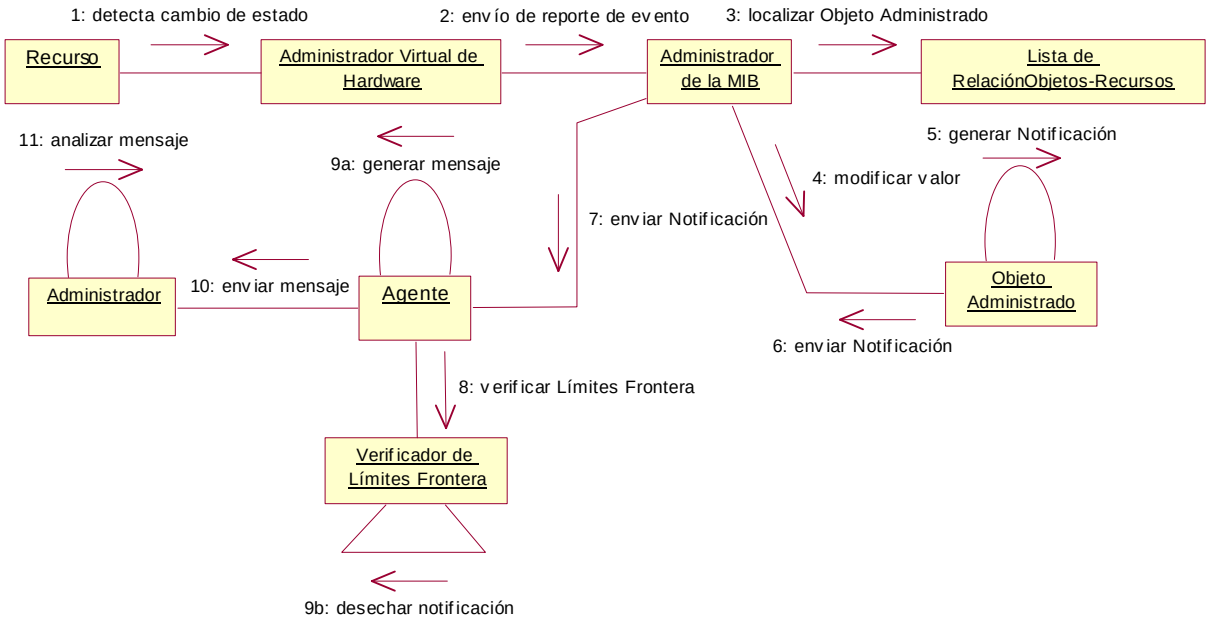
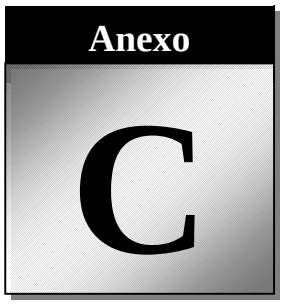


Figura B.15 Diagrama Conceptual Abstracto del Reporte de Evento (Modelo de Internet).



Descripción de Clases del Framework

C.1 Introducción

En el presente anexo se presenta la descripción de las clases definidas en el DDOO, las cuales están agrupadas por componentes de dominio.

Para cada clase, se determina su objetivo, si se trata de una clase concreta, abstracta o si se trata de una interfaz, sus datos miembros, y los métodos que la componen. Los métodos a su vez estarán descritos por su objetivo, sus parámetros de entrada, una especie de pseudocódigo Java, el cual determina la lógica del método, así como el parámetro de salida que le corresponde.

Las descripciones de clases son documentadas en las siguientes secciones.

C.2 Descripción de Clases del Componente "Agente"

C.2.1 AdministratorDirVO

DESCRIPCIÓN DE CLASE		
Clase:	*AdministratorDirVO*	
Tipo:	Concreta	
Objetivo:	Encapsula la dirección IP y puerto de un Administrador.	

MIEMBROS DE CLASE		
Miembro	**Tipo de Dato**	**Objetivo**
ip	String	Dirección IP de un Administrador
port	int	Puerto de entrada de un Administrador

MÉTODOS	
Método:	*AdministratorDirVO(ip: String, port : int)*
Tipo:	Constructor Concreto Público
Objetivo:	Inicializa la dirección IP y el puerto.

Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
ip	String	Dirección IP de un Administrador
port	int	Puerto de entrada de un Administrador

Pseudocódigo

```
this.ip = ip;     // Se asigna la dirección IP del Administrador.
this.port = port; // Se asigna el puerto.
```

Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Ninguno		

Método:	*getIP() : String*
Tipo:	Método Concreto Público
Objetivo:	Obtiene la dirección IP de un Administrador.

Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
Ninguno		

Pseudocódigo

```
return this.ip;   // Se retorna la dirección IP.
```

Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Retorno	String	Dirección IP del Administrador

Método:	*getPort() : int*
Tipo:	Método Concreto Público
Objetivo:	Obtiene el puerto de entrada de un Administrador.

Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
Ninguno		

Pseudocódigo

```
return this.port;   // Se retorna el puerto de entrada.
```

Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Retorno	int	Puerto de entrada del Administrador

C.2.2 Agent

DESCRIPCIÓN DE CLASE

Clase: *Agent*
Tipo: Concreta
Objetivo: Atiende las solicitudes enviadas por el Administrador, generando el servicio correspondiente y ejecutándolo a través de un Proceso.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método: *run() : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Atiende las solicitudes del Administrador.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
AgentResources agentResources = AgentResource.getInstance(); // Recursos del Agente.
ClassesLoader  classesLoader  = ClassesLoader.getInstance(); // Cargador de Clases.

// Se obtienen la cola de mensajes y el administrador de procesos del Administrador.
MessageQueue   messageQueue   = (MessageQueue)
agentResources.getResource(AGENT_MESSAGE_QUEUE);
ProcessManager processManager = (ProcessManager)agentResources.getResource(PROCS_MANAGER);

while(true) {  //Ciclo infinito.
  Message message = messageQueue.getNextMessage(); // Se obtiene el sig. mensaje.
  if (message != null) {               // Si el mensaje no es nulo.
    // Dado el identificador del servicio se intenta obtener una instancia del mismo a
    // través del cargador de clases.
    Service service = classesLoader.getInstance(message.getServiceId());
    if (service != null) {                      // Si se obtuvo instancia.
      Process process = processManager.getProcess(this); // Se obtiene el proceso.
      process.setService(service);                       // Se asigna el serv. al proceso.
      process.notify();                                  // Se reinicia la ejec. del proc.
    }
  }
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.3 AgentResources

DESCRIPCIÓN DE CLASE

Clase: *AgentResources*
Tipo: Concreta
Objetivo: Permite obtener los Recursos o elementos de software que pueden ser utilizados para el cumplimiento de un servicio.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
resources	HashMap	Contiene los recursos disponibles para la ejecución de un Servicio.

MÉTODOS

Método: *AgentResources()*
Tipo: Constructor Concreto Privado
Objetivo: Impide la instanciación directa de esta clase.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Este constructor no tiene código
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getInstance() : AgentResources*
Tipo: Método Concreto Estático Público
Objetivo: Obtiene la única instancia de esta clase.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
if (AgentResources.agentResources.resources == null) { // Verifica que esté inicializado
                                                       // el contenedor de recursos.
  AgentResources.agentResources.resources = new HashMap(); // Inicializa el contenedor.
}
return AgentResouces.agentResources;                   // Retorna la instancia única.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	AgentResources	Instancia única de los Recursos del Agente.

Método: *getResource(name : String) : Object*
Tipo: Método Concreto Público
Objetivo: Obtiene un recurso por su nombre.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
name	String	Nombre del Recurso.

Pseudocódigo

```
return this.resources.get(name); // Obtiene el recurso solicitado.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Object	Recurso solicitado o null si el nombre proporcionado no coincide con alguno de los contenidos.

Método: *setResource(name : String, resource : Object) : void*
Tipo: Método Concreto Público
Objetivo: Almacena un recurso en el contenedor.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
name	String	Nombre del Recurso.
resource	Object	Recurso a ser almacenado.

Pseudocódigo

```
this.resources.put(name, resource); // Se almacena el recurso.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.4 ClassesLoader

DESCRIPCIÓN DE CLASE

Clase: *ClassesLoader*
Tipo: Concreta
Objetivo: Permite obtener dinámicamente objetos. Los objetos pertenecen principalmente a clases abstractas que fueron concretizadas en la configuración del framework.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
classes	HashMap	Contiene los objetos Class.
classesLoader	ClassesLoader	Instancia única de ésta clase.

MÉTODOS

Método: *ClassesLoader()*
Tipo: Constructor Concreto Privado
Objetivo: Impide la instanciación directa de esta clase.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Este constructor no tiene código
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getInstance() : ClassesLoader*
Tipo: Método Concreto Estático Público
Objetivo: Obtiene la única instancia de esta clase.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
if (ClassesLoader.classesLoader.classes == null) {     // Verifica que esté inicializado
                                                       // el contenedor de clases.
 ClassesLoader.classesLoader.classes = new HashMap();  // Inicializa el contenedor.
}
return ClassesLoader.classesLoader;                    // Retorna la instancia única.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ClassesLoader	Instancia única de objetos Class del Agente.

Método: *getObject(key : String) : Object*
Tipo: Método Concreto Público
Objetivo: Obtiene un objeto dada la llave de su clase.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
key	String	Llave de la clase de la cual se quiere instanciar.

Pseudocódigo

```
Class class = this.classes.get(key);
if (class != null) {
  return class.getInstance(key);
} else {
  return null;
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Object	Instancia de clase del objeto solicitado.

Método: *setClass(key : String, className : String) : void*
Tipo: Método Concreto Público
Objetivo: Almacena dentro del contenedor un objeto Class, el cual corresponde a className.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
key	String	Llave de localización de la clase.
className	String	Nombre de la clase.

Pseudocódigo

```
Class class = Class.forName(className);
if (class != null) {
  this.class.put(key, class);
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.5 Filter

DESCRIPCIÓN DE CLASE		
Clase:	*Filter*	
Tipo:	Interfaz	
Objetivo:	Permite discriminar un ojeto para la ejecución de una acción.	
MIEMBROS DE CLASE		
Miembro	**Tipo de Dato**	**Objetivo**
Ninguno		
MÉTODOS		
Método:	*doFilter(object : Object) : Object*	
Tipo:	Método Abstracto Público (*Punto de Variabilidad*)	
Objetivo:	Ejecuta el filtro sobre el objeto a ser discriminado.	
Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
object	Object	Objeto a ser discriminado por el filtro.
Pseudocódigo		
`// La responsabilidad de la implementación de éste método está a cargo del desarrollador // del Agente.`		
Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
object	Object	Objeto resultado de aplicar el filtro. En caso de que no haya sido satisfactoria la aplicación del filtro al Objeto, éste deberá retornar null.

C.2.6 FilterChain

DESCRIPCIÓN DE CLASE

Clase: *FilterChain*
Tipo: Concreta
Objetivo: Elemento que contiene los filtros que se utilizan para discriminar o convertir información antes de que un servicio o método en particular sea ejecutado.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Filters	List	Lista de filtros.

MÉTODOS

Método: *FilterChain()*
Tipo: Constructor Concreto Público
Objetivo: Inicializa la cadena de filtros.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
this.filters = new List();  // Crea la lista que contendrá los filtros.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *addFilter(filter : Filter) : void*
Tipo: Método Concreto Público
Objetivo: Añade un filtro a la cadena.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
filter	Filter	Filtro a ser añadido en la cadena.

Pseudocódigo

```
this.filters.add(filter); // Adición de un filtro a la cadena.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.7 Message

DESCRIPCIÓN DE CLASE

Clase: *Message*
Tipo: Concreta
Objetivo: Encapsula la solicitud que ha efectuado el Administrador, para que se interprete y ejecute un comando en el Agente.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
serviceId	String	Identificador del servicio o comando que debe ejecutarse en el Agente.
messageId	String	Número secuencial que contiene el identificador de mensaje determinado por un Administrador.

MÉTODOS

Método: *getServiceId() : String*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Obtiene el identificador del Servicio que se debe ejecutar en el Agente.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.serviceId; // Retorna el identificador del Servicio.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identifica el servicio al cual corresponde el mensaje.

Método: *getMessageId() : String*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Obtiene el identificador del Mensaje.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.messageId; // Retorna el identificador del Mensaje.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador del mensaje.

Método: *setServiceId(serviceId : String) : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Asigna el identificador del Servicio que se debe ejecutar en el Agente.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
serviceId	String	Identifica el servicio al cual corresponde el mensaje.

Pseudocódigo

```
this.serviceId = serviceId; // Asignación del identificador del Servicio.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *setMessageId(messageId : String) : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Asigna el identificador del Mensaje.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
messageId	String	Identificador del mensaje.

Pseudocódigo

```
this.messageId = messageId; // Asignación del identificador del Mensaje.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.8 MessageCoder

DESCRIPCIÓN DE CLASE		
Clase:	*MessageCoder*	
Tipo:	Abstracta	
Objetivo:	Codifica mensajes y los transforma en un flujo de salida.	
MIEMBROS DE CLASE		
Miembro	**Tipo de Dato**	**Objetivo**
Ninguno		
MÉTODOS		
Método:	*code(message : Message) : OutputStream*	
Tipo:	Método Abstracto Publico (*Punto de Variabilidad*)	
Objetivo:	Permite codificar un mensaje y retornar el flujo de salida correspondiente.	
Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
message	Message	Mensaje a codificar.
Pseudocódigo		
`// La responsabilidad de la implementación de éste método está a cargo del desarrollador // del Agente.`		
Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Retorno	OutputStream	Flujo de salida que pertenece al mensaje codificado o null si el mensaje no puede ser codificado.

C.2.9 MessageDecoder

DESCRIPCIÓN DE CLASE	
Clase:	*MessageDecoder*
Tipo:	Interfaz
Objetivo:	Permite decodificar un flujo de datos al mensaje respectivo enviado por el Administrador.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método:	*decode(stream : InputStream) : Message*
Tipo:	Método Abstracto Público (*Punto de Variabilidad*)
Objetivo:	Decodifica un flujo de entrada para convertirlo en un Mensaje.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
stream	InputStream	Flujo de Datos de donde proviene el mensaje enviado por el Administrador.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Message	Encapsula el mensaje enviado por el Administrador o null si el flujo de datos no corresponde a ningún mensaje válido.

C.2.10 MessageQueue

DESCRIPCIÓN DE CLASE

Clase: *MessageQueue*
Tipo: Abstracta
Objetivo: Contenedor que permite acumular los mensajes enviados por el Administrador o notificaciones de cambio de estado en Recursos del Agente, los cuales serán posteriormente ejecutados de manera concurrente.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
messagesList	List	Lista que contiene los mensajes recibidos por el Agente todavía no interpretados.
filterChain	FilterChain	Cadena de filtros que es aplicada antes de contener el mensaje.

MÉTODOS

Método: *MessageQueue(maxSize : int, filterChain : FilterChain)*
Tipo: Constructor Concreto Público
Objetivo: Inicializa la lista de mensajes, determinando el tamaño máximo de mensajes que puede mantener, así como la cadena de filtros que debe aplicar al intentar insertar un mensaje.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
this.messagesList = new List();  // Crea la lista que contendrá los mensajes.
this.maxSize    = maxSize;     // Se determina la cantidad de mensajes máxima.
this.filterChain  = filterChain; // Se asigna la cadena de filtros que debe aplicarse al
                                 // intentar insertar un mensaje
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *insertMessage(message : Message) : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Inserta un Mensaje en la lista de espera.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
message	Message	Mensaje que va a ser insertado en la lista de espera.

Pseudocódigo

```
message = this.filterChain.processFilters(message)  // Se ejecutan los filtros.
if (message != null) {                  // Si el mensaje no fue desechado por
                                                    // los filtros entonces
  if (this.messagesList.size() < this.maxSize) {    // se verifica que no se haya
                                                    // alcanzado la cantidad máxima de
                                                    // mensajes.
      this.messagesList.add(message);               // Se inserta el mensaje.
  }
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getNextMessage() : Message*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Obtiene el siguiente mensaje de la lista y lo elimina de la misma.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Message	Mensaje siguiente en la lista de espera.

Método: *getMaxSize () : int*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Obtiene el tamaño máximo de la lista de espera.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.maxSize; // Retorna el número máximo de mensajes que pueden ser contenidos.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	int	Número máximo de mensajes permitidos en la lista de espera.

Método: *getSize () : int*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Obtiene el tamaño actual de la lista de espera.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.messagesList.size(); // Retorna el número actual de mensajes en la lista.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	int	Número de mensajes actualmente contenidos en la lista de espera.

C.2.11 MessageReceptor

DESCRIPCIÓN DE CLASE

Clase: *MessageReceptor*
Tipo: Abstracta
Objetivo: Servicio que permite al Agente recibir un flujo de datos, convertirlo a un Mensaje legible por el Agente y si el Mensaje es uno que pueda ser interpretado, lo inserta en la Cola de Mensajes (MessageQueue).

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
port	int	Contiene el puerto por el cual es contactado para recibir la información de una solicitud del Administrador.
decoder	MessageDecoder	Objeto que convierte un flujo de datos en un mensaje que pueda interpretar el Agente.

MÉTODOS

Método: *MessageReceptor(port : int, decoder : MessageDecoder)*
Tipo: Constructor Concreto Público
Objetivo: Inicializa el puerto de escucha al momento de instanciar MessageReceptor. Además mantiene una referencia hacia el decodificador

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
port	int	Número del puerto de entrada.
decoder.	MessageDecoder	Decodificador de la información enviada desde el Administrador hasta el Agente, el cual tranforma el flujo de datos en un Mensaje.

Pseudocódigo

```
this.port    = port;   // Inicializa el puerto de entrada.
this.decoder = decoder;   // Mantiene la referencia hacia decodificador de mensajes.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *run() : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Mantiene la lógica que permite obtener flujos de datos de manera concurrente, y posteriormente convertirlos en mensajes a través del objeto decodificador.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
AgentResources agentResources = AgentResource.getInstance(); // Recursos del Agente.
this.initReceptor(); // Inicializa el receptor de mensajes
messageQueue = (MessageQueue)agentResources.getResource(AGENT_QUEUE);
// Obtiene la Cola de Mensajes a través del AgentResources.
while(true) {       // Ciclo INFINITO
    inputStream  = this.receiveData();              // Recibe el flujo de entrada.
    message      = this.decoder.decode(inputStream); // Se decodifica el mensaje
    messageQueue.insertMessage(message);            // Se inserta el mensaje en la cola.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *initReceptor() : void*
Tipo: Método Abstracto Protegido (*Punto de Variabilidad*)
Objetivo: Inicializa el Receptor de mensajes, poniendo disponible el puerto de entrada, recibiendo ya sea Datagramas o en su defecto Paquetes TCP/IP (Dependiendo de si el protocolo de Administración de Red es Orientado a Conexión o no).

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *destroyReceptor() : void*
Tipo: Método Abstracto Protegido (*Punto de Variabilidad*)
Objetivo: Libera el puerto de entrada y cualesquier elemento utilizado para recibir mensajes.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *receiveData() : InputStream*
Tipo: Método Abstracto Protegido (*Punto de Variabilidad*)
Objetivo: Obtiene un flujo de entrada a través de la tecnología utilizada para recibir datos (orientada a conexión o no orientada a conexión).

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	InputStream	Encapsula el mensaje no codificado por parte del Administrador al Agente.

C.2.12 MessageTransmisor

DESCRIPCIÓN DE CLASE

Clase: *MessageTransmisor*
Tipo: Abstracta
Objetivo: Permite el envío de mensajes de parte del Agente al Administrador.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método: *sendMessage(message : Message, administratorDir : AdministratorDirVO, coder : MessageCoder) : void*
Tipo: Método Concreto Publico (*Punto Común*)
Objetivo: Envía un mensaje al Administrador.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
messageOut	Message	Mensaje a enviar.
administratorDir	AdministratorDirVO	Contiene información del puerto y dirección IP del Administrador.
coder	MessageCoder	Codificador de mensaje.

Pseudocódigo

```
OutputStream stream = coder.code(message); // Obtiene el stream binario del mensaje.
if (stream != null) {                      // Si el stream no es nulo
  this.sendData(stream, administratorDir); // Se envía el mensaje
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *sendData(stream : OutputStream, administratorDir : AdministratorDirVO) : void*
Tipo: Método Abstracto Protegido (*Punto de Variabilidad*)
Objetivo: Envía el mensaje ya codificado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
stream	OutputStream	Mensaje ya codificado.
administratorDir	AdministratorDirVO	Dirección IP y puerto del Administrador

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.2.13 Process

DESCRIPCIÓN DE CLASE

Clase: *Process*
Tipo: Concreta
Objetivo: Permite la ejecución de un servicio solicitado por el Administrador.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
service	Service	Servicio a ser ejecutado por el proceso.
processPool	ProcessManager	Administrador de procesos al cual pertenece el proceso.
processId	int	Identificador del proceso.

MÉTODOS

Método: *Process (processManager : ProcessManager, processId : int)*
Tipo: Constructor Concreto Publico
Objetivo: Inicializa el proceso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
processManager	ProcessManager	Administrador de procesos al cual pertenece el proceso.

Pseudocódigo

```
this.processManager = processManager; // Referencia al Administrador de Procesos.
this.processId = processId;           // Identificador del Proceso
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *run()*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Permite la ejecución de un servicio solicitado por el Administrador.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
while(true) {                  // Ciclo infinito.
  if (this.service != null) {  // Se verifica que el servicio no sea nulo
    this.service.execute();    // Se ejecuta el servicio.
    this.service = null;       // En este punto no es necesario tener ref. A un servicio.
  }
  this.processManager.releaseProcess(this.processId); // Se libera el proceso actual
  this.wait();                 // Se suspende el Thread actual.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *setService(service : Service)*
Tipo: Método Concreto Público
Objetivo: Asigna el Servicio a ser ejecutado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
service	Service	Servicio a ser ejecutado.

Pseudocódigo

```
this.service = service; // Se asigna el servicio
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getProcessId() : int*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador del Proceso

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.processId; // Se retorna el identificador del proceso
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	int	Identificador del Proceso

Método: *getService() : Service*
Tipo: Método Concreto Público
Objetivo: Obtiene el servicio actual del proceso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.service; // Se retorna el servicio.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retono	Service	Servicio actual del proceso.

C.2.14 ProcessManager

DESCRIPCIÓN DE CLASE

Clase: *ProcessManager*
Tipo: Abstracta
Objetivo: Contiene un pool de procesos, los cuales permiten atender a los servicios solicitados por el Administrador.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
processList	List	Contenedor de los procesos.
processesBusy	int	Número de procesos ya ocupados.
size	int	Indica el tamaño del pool de procesos.

MÉTODOS

Método: *ProcessManager(size : int)*
Tipo: Constructor Concreto Publico
Objetivo: Inicializa el pool de procesos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
size	int	Indica el tamaño del pool de procesos.

Pseudocódigo

```
this.processList = new List();            // Se inicializa la lista de procesos.
for(int i=0; i < size; i++) {             // Ciclo de inicialización.
  Process process = new Process(this, i); // Se crea un proceso nuevo.
  this.processList.add(process);          // Se añade el proceso a la lista de procesos.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getSize() : int*
Tipo: Método Concreto Público
Objetivo: Obtiener el tamaño del pool de procesos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.size; // Retorna el tamaño del pool de procesos.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	int	Tamaño del pool de procesos.

Método: *getProcess(agent : Agent) : Process*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Obtiene un proceso del pool de procesos, para ser utilizado en la ejecución de un servicio.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
agent	Agent	Permite que se suspenda el Thread Agente si no es posible obtener un proceso y se reanuda hasta que exista uno disponible.

Pseudocódigo

```
while(this.size <= this.processesBusy); // Mientras que no existan procesos disponibles.
this.processesBusy++;                   // Se aumenta en uno los procesos ocupados.
return this.getProcess();               // Se retorna el proceso.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Process	Encapsula la funcionalidad de un proceso, el cual ejecuta a un servicio.

Método: *releaseProcess(processId : int) : void*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Libera el proceso actual y lo pone disponible para alguna otra solicitud.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
processId	int	Identificador del proceso.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *suspendProcess(processId : int) : void*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Suspende la ejecución de un proceso y lo pone disponible para alguna otra solicitud.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
processId	int	Identificador del proceso.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getProcess() : Process*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Obtiene un proceso del pool de procesos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Process	Proceso obtenido del pool.

C.2.15 Service

DESCRIPCIÓN DE CLASE

Clase: *Service*
Tipo: Abstracta
Objetivo: Contiene las acciones que deben ser realizadas para efecto de cumplir un comando solicitado por un Administrador.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
messageId	String	Identificador del Servicio, el cual corresponde al identificador del comando del protocolo de administración de red a ejecutar.
messageIn	Message	Mensaje de entrada, el cual corresponde a el mensaje enviado por el Administrador para la ejecución del servicio.
messageOut	Message	Mensaje de salida, el cual es el resultado de la ejecución del servicio.

MÉTODOS

Método: *Service(serviceId : String, messageIn : Message)*
Tipo: Constructor Concreto Publico
Objetivo: Inicializa el servicio.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
serviceId	String	Identificador del servicio.
messageIn	Message	Mensaje de entrada.

Pseudocódigo

```
this.serviceId = serviceId;  // Inicializa el identificador del servicio.
this.messageIn = messageIn;  // Inicializa el mensaje de entrada.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *execute()*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Ejecuta el servicio y en caso de que exista un mensaje de salida, entonces se retorna éste mensaje al Administrador .

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
this.messageOut = this.executeCommand();       // Se ejecuta el comando.
AgentResources resources = AgentResources.getInstance();
ClassesLoader  loader    = ClassesLoader.getInstance();
if (this.messageOut != null) {                 // Se verifica si existe mensaje de salida.
  MessageTransmisor transmisor = loader.getObject(MESSAGE_TRANSMISOR); //Transmisor
  MessageCoder      coder = loader.getObject(MESSAGE_CODER); //Codificador de Mssg.
  Vector administrators = resources.getResource(ADMINISTRATORS); // IP de Admons.

  for(int i=0;i < administrators.size();i++) { // Administradores a enviar el mensaje.
    AdministratorDirVO administratorDir = administrators.getElementAt(i);
    transmisor.sendMessage(this.messageOut, coder, administratorDir); // Envía el mensaje.
  }
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *executeCommand() : Message*
Tipo: Método Abstracto Protegido (*Punto de Variabilidad*)
Objetivo: Ejecuta el comando solicitado por el Administrador y genera en su caso el mensaje de salida.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Message	Mensaje resultado de la ejecución del servicio.

Método: *getServiceId() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador del servicio.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.serviceId;
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador del servicio.

Método: *getMessageIn() : Message*
Tipo: Método Concreto Público
Objetivo: Obtiene el mensaje de entrada.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.messageIn;
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Message	Mensaje de entrada.

Método: *getMessageOut() : Message*
Tipo: Método Concreto Público
Objetivo: Obtiene el mensaje de salida.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.messageOut;
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Message	Mensaje de salida.

C.3 Descripción de Clases del Componente "MIB"

C.3.1 MIBContainer

DESCRIPCIÓN DE CLASE	
Clase:	*MIBContainer*
Tipo:	Interfaz
Objetivo:	Contenedor abstracto que permite insertar, eliminar y obtener objetos administrados.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método:	*initialize() : void*
Tipo:	Método Abstracto Público (*Punto de Variabilidad*)
Objetivo:	Inicializa el contenedor de la MIB.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método:	*getManagedObject(managedObjectId : String, instance: String) : ManagedObject*
Tipo:	Método Abstracto Público (*Punto de Variabilidad*)
Objetivo:	Obtiene el objeto administrado solicitado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase del objeto administrado
instance	String	Identificador de instancia de la clase del objeto administrado.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado solicitado o en su defecto null si el objeto administrado no fue encontrado.

Método: *insertManagedObject(managedObject : ManagedObject) : ManagedObject*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Inserta el objeto administrado solicitado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObject	ManagedObject	Objeto Administrado a ser insertado.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado insertado o en su defecto null si el objeto administrado no fué insertado.

Método: *deleteManagedObject(managedObjectId : String, instance : String) : ManagedObject*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Elimina en el contenedor el objeto administrado solicitado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase del objeto administrado
instance	String	Identificador de instancia de la clase del objeto administrado.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado eliminado o en su defecto null si el objeto administrado no fué eliminado.

C.3.2 MIBManager

DESCRIPCIÓN DE CLASE

Clase: *MIBManager*
Tipo: Concreta
Objetivo: Administrador de la MIB, el cual es el punto de entrada para la actualización y manipulación de Objetos Administrados.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
container	MIBContainer	Contenedor de Objetos Administrados.

MÉTODOS

Método: *MIBManager(container : MIBContainer)*
Tipo: Constructor Concreto Público
Objetivo: Inicializa el administrador de la MIB y asigna el contenedor de Objetos Administrados.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
container	MIBContainer	Contenedor de la MIB

Pseudocódigo

```
this.container = container;   // Se asigna el contenedor de la MIB ya inicializado.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *run() : void*
Tipo: Método Concreto Público (Punto Común)
Objetivo: Atiende los mensajes de cambio de estado enviados por el Administrador Virtual de Hardware.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
AgentResources agentResources = AgentResource.getInstance(); // Recursos del Agente.

// Se obtiene la cola de mensajes de la MIB y del Agente.
MessageQueue   mibMessageQueue   = (MessageQueue)
AgentResources.getResource(MIB_MESSAGE_QUEUE);
MessageQueue   agentMessageQueue   = (MessageQueue)
AgentResources.getResource(AGENT_MESSAGE_QUEUE);

while(true) {  //Ciclo infinito.
  ReportMessage message = (ReportMessage)messageQueue.getNextMessage(); // Sig. mensaje.
  if (message != null) {              // Si el mensaje no es nulo.
                                                  // Se asigna el valor enviado por AVH.
    ManagedObject managedObject = this.setManagedObject(message.getObjectId(),
    message.getInstance(), message.getValue());
    if (managedObject != null) {                // Si se asignó el valor del OA.
    // Se inserta el mensaje de reporte en la cola del Agente para que a través de
    // un servicio se filtre el mensaje y en su caso se envíe al Administrador.
      agentMessageQueue.insertMessage(message);
} } }
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getManagedObject(managedObjectId : String, instance: String) : ManagedObject*
Tipo: Método Concreto Público (Punto Común)
Objetivo: Obtiene el objeto administrado solicitado a través del contenedor. Además verifica las reglas de acceso del objeto.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase del objeto administrado
instance	String	Identificador de instancia de la clase del objeto administrado.

Pseudocódigo

```
// Se obtiene el objeto administrado del contenedor.
ManagedObject managedObject = this.cantainer.getManagedObject(managedObjectId, instance);
if (managedObject != null) {             // Si el objeto fué encontrado.
  if (managedObject.getRead() == true) { // Se verifica que tenga permiso de lectura.
    return managedObject;                // Se envía el Objeto Administrado.
  } else {
    return null;                         // Se retorna null por permiso denegado.
  }
} else {
   return null;                          // Se retorna null por no encontrar OA.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado solicitado o en su defecto null si el objeto administrado no cumple con el permiso de lectura o no fue encontrado.

Método: *setManagedObject(managedObjectId : String, instance : String, value : Object) : ManagedObject*
Tipo: Método Concreto Público (Punto Común)
Objetivo: Obtiene el objeto administrado solicitado a través del contenedor y asigna cierto valor al objeto. Además verifica las reglas de acceso del objeto para determinar si se debe o nó asignar el valor enviado como parámetro.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase del objeto administrado
instance	String	Identificador de instancia de la clase del objeto administrado.
value	Object	Valor que debe ser asignado al objeto administrado.

Pseudocódigo

```
// Se obtiene el objeto administrado del contenedor.
ManagedObject managedObject = this.container.getManagedObject(managedObjectId, instance);
if (managedObject != null) {               // Si el objeto fué encontrado.
  if (managedObject.getWrite() == true) {  // Se verifica que tenga permiso de escritura.
    managedObject.setValue();              // Se asigna el valor enviado.
    return managedObject;                  // Se retorna el objeto administrado.
  } else {
    return null;                           // Se retorna null por permiso denegado.
  }
} else {
   return null;                            // Se retorna null por no encontrar OA.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado cuyo valor ha sido modificado o en su defecto null si el objeto administrado no cumple con el permiso de escritura o no fue encontrado.

Método: *insertManagedObject(managedObject : ManagedObject) : ManagedObject*
Tipo: Método Concreto Público (Punto Común)
Objetivo: Inserta en el contenedor el objeto administrado solicitado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObject	ManagedObject	Objeto Administrado a ser insertado en el contenedor.

Pseudocódigo

```
// Se inserta el objeto administrado y se retorna.
return this.container.insertManagedObject(managedObject);
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado insertado en el contenedor o en su defecto null si el objeto administrado no fué insertado.

Método: *deleteManagedObject(managedObjectId : String, instance : String) : ManagedObject*
Tipo: Método Concreto Público (Punto Común)
Objetivo: Elimina en el contenedor el objeto administrado solicitado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase del objeto administrado
instance	String	Identificador de instancia de la clase del objeto administrado.

Pseudocódigo

```
// Se elimina el objeto administrado y se retorna.
return this.container.deleteManagedObject(managedObjectId, instance);
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ManagedObject	Objeto Administrado eliminado en el contenedor o en su defecto null si el objeto administrado no fué eliminado.

C.3.3 ManagedObject

DESCRIPCIÓN DE CLASE

Clase: *ManagedObject*
Tipo: Abstracta
Objetivo: Encapsula información referente a una instancia de Objeto Administrado.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
id	String	Identificador de clase de Objeto Administrado.
instance	String	Identificador de instancia de Objeto Administrado.
read	boolean	Permiso de lectura.
write	boolean	Permiso de escritura.

MÉTODOS

Método: *ManagedObject(id : String, instance : String, read : boolean, write : boolean)*
Tipo: Constructor Concreto Público
Objetivo: Instancía a un Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
id	String	Identificador de clase de Objeto Administrado.
instance	String	Identificador de instancia de Objeto Administrado.
read	boolean	Permiso de lectura.
write	boolean	Permiso de escritura.

Pseudocódigo

```
this.id       = id;       // Se asigna el identificador de clase del OA.
this.instance = instance; // Se asigna el identificador de instancia del OA.
this.read     = read;     // Se asigna el permiso de lectura del OA.
this.write    = write;    // Se asigna el permiso de escritura del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getId() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de clase del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.id; // Se retorna el identificador de clase del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de clase de Objeto Administrado.

Método: *getInstance() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de instancia del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.instance; // Se retorna el identificador de instancia del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de instancia de Objeto Administrado.

Método: *getRead() : boolean*
Tipo: Método Concreto Público
Objetivo: Obtiene el permiso de lectura del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.read; // Se retorna el permiso de lectura del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Permiso de lectura del Objeto Administrado.

Método: *getWrite() : boolean*
Tipo: Método Concreto Público
Objetivo: Obtiene el permiso de escritura del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.write; // Se retorna el permiso de escritura del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Permiso de escritura del Objeto Administrado.

Método: *setValue(value : Object) : void*
Tipo: Método Abstracto Público Sincronizado (*Punto de Variabilidad*)
Objetivo: Asigna el valor correspondiente del Objeto Administrado de una forma abstracta y sincronizada, para de ésta forma impedir que dos procesos hagan modificaciones concurrentes del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
value	Object	Valor abstracto del Objeto Administrado.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getValue() : Object*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Obtiene el valor abstracto del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Object	Valor abstracto del Objeto Administrado

Método: *getState() : boolean*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Identifica si el estado del Objeto Administrado ha sobrepasado sus límites frontera.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Se retorna true si el estado del objeto administrado es normal y false de lo contrario.

C.4 Descripción de Clases del Componente "AVH"

C.4.1 AVHManager

DESCRIPCIÓN DE CLASE

Clase: *AVHManager*
Tipo: Abstracta
Objetivo: Administrador Virtual de Hardware, el cual monitorea constantemente los recursos del Elemento de Red.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
container	ResourcesContainer	Contenedor de Recursos de Elemento de Red.

MÉTODOS

Método: *AVHManager(container : ResourcesContainer)*
Tipo: Constructor Concreto Público
Objetivo: Inicializa el Administrador Virtual de Hardware y asigna el contenedor de Recursos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
container	ResourcesContainer	Contenedor de Recursos del Elemento de Red.

Pseudocódigo

```
this.container = container;   // Se asigna el contenedor de recursos ya inicializado.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *run() : void*
Tipo: Método Concreto Público (*Punto Común*)
Objetivo: Monitorea los recursos del elemento de red y en caso de que el estado de un Recurso haya cambiado, entonces se envía al Administrador de la MIB un reporte.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Se obtienen los recursos del Agente, el manejador de clases, la cola de mensajes del
// Agente, el creador de reportes de cambio de estado,  el intervalo de tiempo entre cada
// monitoreo de recursos y el Iterador de Recursos.
AgentResources    agentResources = AgentResource.getInstance();
ClassesLoader     classesLoader  = ClassesLoader.getInstance();
MessageQueue      messageQueue   = agentResources.getResource(MIB_MESSAGE_QUEUE);
ReportCreator     reportCreator  = (ReportCreator)classesLoader.getObject(REPORT_CREATOR);
int               sleepTime      = agentResources.SLEEP_TIME;
ResourcesIterator iterator       = this.container.getIterator();
while(true) {                                       // Ciclo infinito.
  Resource resource = iterator.next();              // Siguiente Recurso a monitorear.
  boolean resourceState = resource.getState();      // Estado del recurso.
  if (resourceState == false) {                     // Si estado del recurso no es normal.
    ReportMessage reportMessage = reportCreator.createMessageReport(resource); //Reporte.
    // Se inserta el reporte en la cola de mensajes de la MIB
    messageQueue.insertMessage(reportMessage);
  }
  this.sleep(sleepTime); // Se interrumpe el proceso de monitoreo por sleepTime miliseg.
}
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getResourcesContainer() : ResourcesContainer*
Tipo: Método Concreto Público
Objetivo: Obtiene el contenedor del Contenedor de Recursos del Elemento de Red.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.resourcesContainer; // Retorna el contenedor de recursos del Elemento de Red.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ResourcesContainer	Encapsula el contenedor de recursos.

Método: *createMessageReport(resource : Resource) : MessageReport*
Tipo: Método Concreto Público
Objetivo: Crea un mensaje de reporte de cambio de estado en uno de los Recursos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
resource	Resource	Recurso para el cual se va a generar un reporte de cambio de estado

Pseudocódigo

```
AgentResources    agentResources = AgentResource.getInstance(); // Contenedor de Recursos
// Se obtiene del contenedor de recursos el creador de reportes de cambio de estado.
ReportCreator reportCreator = (ReportCreator)agentResources.getResource(REPORT_CREATOR);
return reportCreator.createReport(resource);
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	MessageReport	Reporte de cambio de estado de un Recurso de Elemento de Red.

C.4.2 ReportCreator

DESCRIPCIÓN DE CLASE	
Clase:	*ReportCretor*
Tipo:	Abstracta
Objetivo:	Genera un reporte de cambio de estado en un Recurso.

MIEMBROS DE CLASE		
Miembro	**Tipo de Dato**	**Objetivo**
Ninguno		

MÉTODOS	
Método:	*createReport(resource : Resource) : ReportMessage*
Tipo:	Método Abstracto Público (*Punto de Variabilidad*)
Objetivo:	Genera un reporte de cambio de estado de un Recurso.

Parámetros de Entrada		
Parámetro	**Tipo de Dato**	**Objetivo**
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Ninguno		

C.4.3 ReportMessage

DESCRIPCIÓN DE CLASE

Clase: *ReportMessage;* extiende de *Message*
Tipo: Concreta
Objetivo: Encapsula el reporte de cambio de estado de un Recurso u Objeto Administrado.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
objectId	String	Identificador de clase del Objeto Administrado del cual proviene el Reporte.
instance	String	Identificador de instancia del Objeto Administrado del cual proviene el Reporte.
value	Object	Estado del Objeto Administrado o en su defecto estado del Recurso.

MÉTODOS

Método: *ReportMessage(objectId : String, instance : String, value : Object)*
Tipo: Constructor Concreto Público
Objetivo: Inicializa el reporte de cambio de estado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
objectId	String	Identificador de clase del Objeto Administrado del cual proviene el Reporte.
instance	String	Identificador de instancia del Objeto Administrado del cual proviene el Reporte.
value	Object	Estado del Objeto Administrado o en su defecto estado del Recurso.

Pseudocódigo

```
this.objectId = objectId; // Se asigna el identificador de clase del OA.
this.instance = instance; // Se asigna el identificador de instancia del OA.
this.value    = value;    // Se asigna el estado del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getObjectId() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de clase del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.objectId; // Se retorna el identificador de clase del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de clase de Objeto Administrado.

Método: *getInstance() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de instancia del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.instance; // Se retorna el identificador de instancia del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de instancia de Objeto Administrado.

Método: *getValue() : Object*
Tipo: Método Concreto Público
Objetivo: Obtiene el valor abstracto del Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.value; // Se retorna el estado del OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Object	Valor abstracto del Objeto Administrado

C.4.4 Resource

DESCRIPCIÓN DE CLASE

Clase: *Resource*
Tipo: Abstracta
Objetivo: Encapsula información referente a un Recurso de Elemento de Red.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
id	String	Identificador del Recurso.
managedObjectId	String	Identificador de la clase del Objeto Administrado relacionado con el Recurso.
managedObject Instance	String	Identificador de instancia del Objeto Administrado relacionado con el Recurso.

MÉTODOS

Método: *ManagedObject(id : String, managedObjectId : String, managedObjectInstance : String)*
Tipo: Constructor Concreto Público
Objetivo: Instancía a un Recurso de Elemento de Red.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
id	String	Identificador del Recurso.
managedObjectId	String	Identificador de la clase del Objeto Administrado relacionado con el Recurso.
managedObject Instance	String	Identificador de instancia del Objeto Administrado relacionado con el Recurso.

Pseudocódigo

```
this.id = id;                          // Se asigna el identificador del Recurso.
this.managedObjectId = managedObjectId; // Id. de clase del OA relacionado con el Recurso.
this.managedObjectInstance = managedObjectInstance; // Id. de instancia del OA rel. Rec.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getId() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador del Recurso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.id; // Se retorna el identificador del Recurso.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador del Recurso.

Método: *getManagedObjectId() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de clase del Objeto Administrado relacionado con el Recurso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.managedObjectId; // Se retorna el identificador de clase del OA
                             // relacionado con el recurso
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de clase del Objeto Administrado relacionado con el Recurso.

Método: *getManagedObjectInstance() : String*
Tipo: Método Concreto Público
Objetivo: Obtiene el identificador de instancia del Objeto Administrado relacionado con el Recurso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
return this.managedObjectInstance; // Se retorna el identificador de instancia del OA
                                   // relacionado con el recurso.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	String	Identificador de instancia del Objeto Administrado relacionado con el Recurso.

Método: *setValue(value : Object) : void*
Tipo: Método Abstracto Público Sincronizado (*Punto de Variabilidad*)
Objetivo: Asigna el valor correspondiente del Recurso de una forma abstracta y sincronizada, para de ésta forma impedir que dos procesos hagan modificaciones concurrentes del valor del Recurso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
value	Object	Valor abstracto del Objeto Administrado.

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *getValue() : Object*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Obtiene el valor abstracto del Recurso.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Object	Valor abstracto del Recurso.

Método: *getState() : boolean*
Tipo: Método Abstracto Público (*Punto de Variabilidad*)
Objetivo: Identifica si el estado del Recurso es normal o está fuera de lo normal.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Se retorna true si el estado del recurso es normal y false de lo contrario.

C.4.5 ResourcesContainer

DESCRIPCIÓN DE CLASE	
Clase:	*ResourcesContainer*
Tipo:	Abstracta
Objetivo:	Contenedor de Recursos del Elemento de Red.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
resourcesList	List	Lista de Recursos del Elemento de Red
resourceMapping	HashMap	Mapea el identificador del Recurso en el recurso en sí.
resourcesMIB Mapping	HashMap	Mapea el identificador de clase y la instancia del Objeto Administrado por el Recurso del elemento de red correspondiente.

MÉTODOS

Método:	*initialize() : void*
Tipo:	Método Asbtracto Público (*Punto de Variabilidad*)
Objetivo:	Inicializa el contenedor de Recursos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método:	*getResource(id : String) : Resource*
Tipo:	Método Concreto Público
Objetivo:	Obtiene un Recurso dado un identificador.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
id	String	Identificador del Recurso

Pseudocódigo

```
return (Resource)this.resourcesMapping.get(id); // Se retorna el Recurso dado el ID.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Resource	Recurso solicitado dado un identificador o null si no existe un Recurso que cuente con el idetificador.

Método: *getResource(managedObjectId : String, managedObjectInstance : String) : Resource*
Tipo: Método Concreto Público
Objetivo: Obtiene un Recurso dado el identificador de clase e identificador de instancia de un Objeto Administrado.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase de un Objeto Administrado.
ManagedObject Instance	String	Identificador de instancia de un Objeto Administrado.

Pseudocódigo

```
return (Resource)this.resourcesMIBMapping.get(managedObjectId + "_" +
                                              managedObject);
// Se retorna el Recurso dado el identificador de clase e instancia de un OA.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Resource	Recurso solicitado dado un identificador o null si no existe un Recurso que cuente con el idetificador.

Método: *insertResource(resource : Resource) : void*
Tipo: Método Concreto Público
Objetivo: Inserta un recurso en la lista.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
managedObjectId	String	Identificador de clase de un Objeto Administrado.
ManagedObject Instance	String	Identificador de instancia de un Objeto Administrado.

Pseudocódigo

```
this.resourcesList.add(resource);                    // Se añade el recurso a la lista.
this.resourcesMapping.add(resource.getId(), resource); // Se añade el recurso al mapeo.
                                                     // Se añade el recurso al mapeo de MIB.
this.resourcesMIBMapping.add(resource.getManagedObjectId() + "_" +
                        resource.getManagedObjectInstance());
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *deleteResource(id : String) : Resource*
Tipo: Método Concreto Público
Objetivo: Elimina un recurso del contenedor.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
id	String	Identificador del Recurso.

Pseudocódigo

```
Resource resource = this.resourcesMapping.get(id); // Se obtiene el recurso dado su ID.
if (resource != null) {                            // Si no es null.
  // Se remueve el recurso del mapeo, el mapeo de MIB y de la lista de recursos.
  this.resourcesMapping.remove(id);
  this.resourcesMIBMapping.remove(resource.getManagedObjectId() + "_" +
                                  resource.getManagedObjectInstance());
  this.resourcesList.remove(resource);
}
return resource;                                   // Se retorna el recurso eliminado.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Resource	Recurso eliminado del contenedor o null si el ID del recurso no existe.

Método: *getResourcesIterator() : ResourcesIterator*
Tipo: Método Concreto Público
Objetivo: Obtiene un iterador de recursos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Se obtiene el cargador de clases, los recursos del Agente, el objeto class que
// corresponde a la clase del iterador de recursos y por último se instancía el iterador.
ClassesLoader loader = ClassesLoader.getInstance();
AgentResources agentResources = AgentResources.getInstance();
Class iteratorClass = loader.getClass(agentResources.RESOURCES_ITERATOR);
ResourcesIterator iterator = (ResourcesIterator)
                             iteratorClass.newInstance(this.resourcesList);
return iterator; // Se retorna el iterador.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	ResourcesIterator	Iterador de recursos.

Método: *getNext() : Resource*
Tipo: Método Asbtracto Público (*Punto de Variabilidad*)
Objetivo: Obtiene el siguiente recurso a ser monitoreado por el Administrador Virtual de Hardware.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// La responsabilidad de la implementación de éste método está a cargo del desarrollador
// del Agente.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Resource	Recurso siguiente en el iterador.

C.4.6 ResourcesIterator

DESCRIPCIÓN DE CLASE	
Clase:	*ResourcesIterator*
Tipo:	Abstracta
Objetivo:	Iterador de Recursos del Elemento de Red que sirve como auxiliar en el monitoreo de los mismos.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
resourcesList	List	Lista de Recursos del Elemento de Red

MÉTODOS

Método:	*ResourcesIterator(resourcesList : List)*
Tipo:	Constructor Concreto Público.
Objetivo:	Inicializa el iterador de recursos, recibiendo la lista sobre la cual va a iterar.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
resourcesList	List	Lista de Recursos del Elemento de Red

Pseudocódigo

```
this.resourcesList = resourcesList; // Se asigna la lista de recursos al iterador.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.5 Descripción de Clases del Configurador del Framework

C.5.1 AgentExecuter

DESCRIPCIÓN DE CLASE

Clase: *AgentExecuter*
Tipo: Concreta
Objetivo: Inicializa el Agente de Administración de Red basándose en el archivo de configuración XML.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método: *main(args : String[]) : void*
Tipo: Método Concreto Público Estático
Objetivo: Inicializa el Agente de Administración de Red dada la ruta del archivo XML de configuración.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
args	String []	Argumentos de inicialización, donde debe enviarse el nombre del archivo XML el cual contiene la configuración del framework.

Pseudocódigo

```
String xmlFilePath = args[0]; // Se obtiene la ruta del Archivo XML de configuración.
if (xmlFilePath == null) {    // Si no se determina la ruta del archivo XML.
   System.out.println("Se debe proporcionar la ruta del archivo XML de configuración");
   return;                 // Se despliega el mensaje de error y se sale de la aplicación.
}
XMLFactory xmlFactory = new XMLFactory(); // Se crea la fábrica de datos de XML.
File       file = new File(xmlFilePath);  // Se obtiene el archivo de configuración XML.
Vector     data;
if (file.exists() == false) {             // Si el archivo de configuración no existe
   System.out.println("La ruta del archivo XML de configuración no existe.");
   return;                 // Se despliega el mensaje de error y se sale de la aplicación.
}
                  // Se obtiene el flujo de entrada a partir del archivo de configuración.
FileInputStram fileInputStream = new FileInputStream(file);

// Se convierten los datos XML en recursos y clases del Agente.
xmlFactory.createXMLData(fileInputStream,Class.forName(
                        "com.networkagent.framework.config.ConfigXMLHelper"));
// Se obtiene la instancia que almacena los recursos del Agente.
AgentResources agentResources = AgentResources.getInstance();
Agent agent = new Agent();               // Se crea una instancia del Agente.
MIBManager mibManager = new MIBManager(  // Se crea una instancia del MIBManager
                            agentResources.getResource(AgentResourcesKeys.MIB_CONTAINER));
AVHManager avhManager = new AVHManager(  // Se crea una instancia del AVHManager
                       agentResources.getResource(AgentResourcesKeys.RESOURCES_CONTAINER));
// Se ponen disponibles como recursos instancias de Agent, MIBManager y AVHManager.
agentResources.setResource(AgentResourcesKeys.AGENT, agent);
agentResources.setResource(AgentResourcesKeys.MIB_MANAGER, mibManager);
agentResources.setResource(AgentResourcesKeys.AVH_MANAGER, avhManager);
// Se ejecutan los Threads Agent, MIBManager, AVHManager y MessageReceptor.
new Thread(agent).start();
new Thread(mibManager).start();
new Thread(avhManager).start();
new Thread(agentResources.getResource(AgentResourcesKeys.MESSAGE_RECEPTOR)).start();
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

C.5.2 ConfigXMLHelper

DESCRIPCIÓN DE CLASE

Clase: *ConfigXMLHelper*; implementa a *XMLIzable*
Tipo: Concreta
Objetivo: Analiza etiquetas de XML y su contenido para obtener los elementos configurados del framework y ponerlos disponibles para su uso en *AgentResources* o en *ClassesLoader*.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
agentResources	AgentResources	Contenedor de recursos de software del Agente.
classesLoader	ClassesLoader	Contenedor de objetos Class, los cuales son auxiliares para realizar el enlace dinámico en el framework.
before	String	Contiene el nombre del tag actual que se está analizando.
data	Vector	Vector que contiene el resultado del análisis XML.

MÉTODOS

Método: *ConfigXMLHelper()*
Tipo: Constructor Concreto Público
Objetivo: Inicializa el analizador de XML.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Se obtiene la instancia de AgentResources, ClassesLoader y se inicializa el Vector de
// datos de retorno
this.agentResources = AgentResources.getInstance();
this.classesLoader  = ClassesLoader.getInstance();
this.data           = new Vector();
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Ninguno		

Método: *setAttribute(elementName : String, attrs : Attributes) : boolean*
Tipo: Método Concreto Público
Objetivo: Dado una etiqueta de XML obtiene el contenido de sus atributos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
elementName	String	Nombre de la etiqueta XML de la cual se van a obtener sus atributos.
attrs	Attributes	Contiene el valor de los atributos de un tag en específico.

Pseudocódigo

```
if (elementName.equals(MESSAGE_DECODER) == true) {
   Class messageDecoder = Class.forName(attrs.getValue(0));
   this.classesLoader.setClass(MESSAGE_DECODER, messageDecoder);
} else if (elementName.equals(MESSAGE_DECODER)) {
   Constructor messageReceptor = Class.forName(attrs.getValue(0)).getConstructors()[0];
   Object[] parameters = new Object[2];
   parameters[0] = this.classesLoader.getObject(MESSAGE_DECODER);
   parameters[1] = new Integer(attrs.getValue(1));
   this.agentResources.setResource(MESSAGE_DECODER,
                                   messageReceptor.newInstance(parameters));
} else if (elementName.equals(AGENT_FILTER_CHAIN)) {
   this.filterChain = new FilterChain();
   this.agentResources.setResource(AGENT_FILTER_CHAIN);
} else if (elementName.equals(FILTER)) {
   this.filterChain.addFilter(Class.forName(attrs.getValue(0)));
}
```

```
  else if (elementName.equals(AGENT_QUEUE)) {
   Constructor agentQueue = Class.forName(attrs.getValue(0)).getConstructors()[0];
   Object[] parameters = new Object[2];
   parameters[0] = new Integer(attrs.getValue(1));
   parameters[1] = this.classesLoader.getObject(AGENT_FILTER_CHAIN);
   this.agentResources.setResource(AGENT_QUEUE,
                                   agentQueue.newInstance(parameters));
} else if (elementName.equals(PROCESS_MANAGER)) {
   Constructor processManager = Class.forName(attrs.getValue(0)).getConstructors()[0];
   Object[] parameters = new Object[1];
   parameters[0] = new Integer(attrs.getValue(1));
   this.agentResources.setResource(PROCESS_MANAGER,
                                   processManager.newInstance(parameters));
} else if (elementName.equals(SERVICE)) {
   this.agentResources.setResource(attrs.getValue(0),
                                   Class.forName(attrs.getValue(1)).newInstance());
} else if (elementName.equals(MESSAGE_TRANSMISOR)) {
   this.classesLoader.setClass(MESSAGE_TRANSMISOR,
                               Class.forName(attrs.getValue(0)).newInstance());
} else if (elementName.equals(MESSAGE_CODER))
   this.classesLoader.setClass(MESSAGE_CODER,
                               Class.forName(attrs.getValue(0)).newInstance());
} else if (elementName.equals(ADMINISTRATORS)) {
   this.agentResources.setResource(ADMINISTRATORS, new Vector());
} else if (elementName.equals(ADMINISTRATOR)) {
   Vector administrators = this.agentResources.getResource(ADMINISTRATOR);
   AdministratorVO administrator = new AdministratorVO
                                  (attrs.getValue(0),Integer.parseInt(attrs.getValue(1)));
   administrators.add(administrator);
} else if (elementName.equals(RESOURCE)) {
   this.agentResources.setResource(attrs.getValue(0),
                                   Class.forName(attrs.getValue(1)).newInstance());
} else if (elementName.equals(ADD_CLASS)) {
   this.agentResources.setResource(attrs.getValue(0),
                                   Class.forName(attrs.getValue(1)));
} else if (elementName.equals(MIB_CONTAINER)) {
   this.agentResources.setResource(MIB_CONTAINER,
                                   Class.forName(attrs.getValue(0)).newInstance());
} else if (elementName.equals(MIB_FILTER_CHAIN)) {
   this.filterChain = new FilterChain();
   this.agentResources.setResource(MIB_FILTER_CHAIN);
} else if (elementName.equals(MIB_QUEUE)) {
   Constructor mibQueue = Class.forName(attrs.getValue(0)).getConstructors()[0];
   Object[] parameters = new Object[2];
   parameters[0] = new Integer(attrs.getValue(1));
   parameters[1] = this.classesLoader.getObject(MIB_FILTER_CHAIN);
   this.agentResources.setResource(MIB_QUEUE,
                                   agentQueue.newInstance(parameters));
} else if (elementName.equals(AVH_CONTAINER)) {
   this.agentResources.setResource(AVH_CONTAINER,
                                   Class.forName(attrs.getValue(0)).newInstance());
} else if (elementName.equals(RESOURCES_ITERATOR)) {
   this.agentResources.setResource(RESOURCES_ITERATOR,
                                   Class.forName(attrs.getValue(0)).newInstance());
} else if (elementName.equals(REPORT_CREATOR)) {
   this.agentResources.setResource(REPORT_CREATOR,
                                   Class.forName(attrs.getValue(0)).newInstance());
}
this.before = elementName;

return true;
```

Parámetro de Salida		
Parámetro	**Tipo de Dato**	**Objetivo**
Retorno	boolean	Indica si el análisis de etiquetas fue realizado correctamente.

Nota: Los métodos restantes que implementan a la interfaz XMLIzable están vacíos y no es relevante que se describan. Sólo retornan valores default.

C.5.3 XMLFactory

DESCRIPCIÓN DE CLASE	
Clase:	*XMLFactory*
Tipo:	Concreta
Objetivo:	Contiene la lógica suficiente para convertir un flujo XML en un Vector que contiene objetos representativos o resultantes del XML.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
xmlIzable	XMLIzable	Objeto que analiza el contenido de cada tag de XML y que convierte la información XML a Objetos.

MÉTODOS

Método: *createXMLData(stream : InputStream, xmlClass : Class) : Vector*
Tipo: Método Concreto Público
Objetivo: Contiene la lógica para convertir un flujo XML a un Vector de objetos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
stream	InputStream	Flujo XML.
xmlClass	Class	Objeto Class que encapsula a la clase de la cual se instancia para analizar el contenido XML y convertir la información XML en Objetos.

Pseudocódigo

```
this.xmlIzable = xmlClass.newInstance() // Se crea instancia de la clase que analiza XML.
SAXParser parser = new SAXParser();     // Se crea un parser de XML, el cual utiliza la
                                        // instancia de la clase que analiza XML.
parse.parse(stream, this.xmlIzable);    // Se parsea el flujo XML delegando la lógica de
                                        // interpretación y análisis XML
                                        // a la instancia xmlIzable.
return xmlIzable.getData();             // Se retorna el Vector de Objetos que fueron
                                        // obtenidos del flujo XML.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Vector	Vector de Objetos los cuales fueron obtenidos del análisis del XML.

C.5.4 XMLIzable

DESCRIPCIÓN DE CLASE	
Clase:	*XMLIzable*
Tipo:	Interfaz
Objetivo:	Contiene los métodos que se deben implementar para que el parser de XML delegue el análisis de las etiquetas a la clase que implemente a esta interfaz.

MIEMBROS DE CLASE

Miembro	Tipo de Dato	Objetivo
Ninguno		

MÉTODOS

Método: *setAttribute(elementName : String, attrs : Attributes) : boolean*
Tipo: Método Abstracto Público
Objetivo: Dado una etiqueta de XML obtiene el contenido de sus atributos.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
elementName	String	Nombre de la etiqueta XML de la cual se van a obtener sus atributos.
attrs	Attributes	Contiene el valor de los atributos de un tag en específico.

Pseudocódigo

```
// Método Abstracto.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Indica si el análisis de etiquetas fue realizado correctamente.

Método: *setAttribute(value : String) : boolean*
Tipo: Método Abstracto Público
Objetivo: Dado una etiqueta de XML obtiene el contenido de la misma.

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
value	String	Valor que tiene una etiqueta de XML.

Pseudocódigo

```
// Método Abstracto.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Indica si el análisis de etiquetas fue realizado correctamente.

Método: *signal(elementName : String) : boolean*
Tipo: Método Abstracto Público
Objetivo: Identifica cuando se cierra una etiqueta de XML

Parámetros de Entrada

Parámetro	Tipo de Dato	Objetivo
elementName	String	Nombre de la etiqueta XML de la cual se van a obtener sus atributos.

Pseudocódigo

```
// Método Abstracto.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	boolean	Indica si el análisis de etiquetas fue realizado correctamente.

Método: *getData() : Vector*
Tipo: Método Abstracto Público
Objetivo: Retorna el Vector de datos resultado del análisis de XML

Parámetros de Entrada

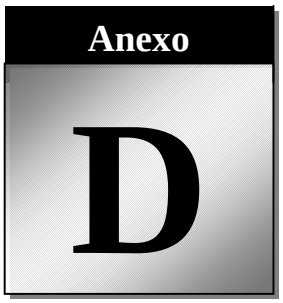
Parámetro	Tipo de Dato	Objetivo
Ninguno		

Pseudocódigo

```
// Método Abstracto.
```

Parámetro de Salida

Parámetro	Tipo de Dato	Objetivo
Retorno	Vector	Vector de datos que es el resultado del análisis de XML.



Implementación del Agente Proxy SNMP para el Ruteador Cisco 2501

D.1 Introducción

En el presente anexo se presenta la implementación de las clases definidas en la aplicación Agente Proxy del Ruteador Cisco 2501.

Cabe destacar que estas clases se encuentran definidas en la aplicación del framework de dominio (Véase el "*Capítulo 8. Aplicación del Framework: Agente Proxy SNMP para el Ruteador Cisco 2501*"), en la cual se valida el análisis, diseño e implementación del framework.

D.2 Implementación de la Capa "Agente"

D.2.1 ConfigXMLHelper

```
/**
 * @(#)com.networkagent.ciscorouter.agent.CommunityFilter.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  28 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Filtra mensajes GET-REQUEST de acuerdo a la comunidad del Agente.
* Si lagún mensaje no pertenece a la comunidad del Agente, este es rechazado.
*/

import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.framework.util.Filter;
import com.networkagent.ciscorouter.agent.SNMPGetMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;
import com.networkagent.ciscorouter.mib.MIBConstants;
```

```
public class CommunityFilter implements Filter
{
   public Object doFilter(Object object) {
      Message message;

      message = (Message)object;
      if (message.getServiceId() == Integer.toHexString(SNMPConstants.SNMP_GET)) {
         SNMPGetMessage getMessage;

         getMessage = (SNMPGetMessage) message;
         if (!getMessage.getCommunityName().equals(SNMPConstants.COMMUNITY_NAME)) {
            return null;
         }
      }

      return message;
   }
}
```

D.2.2 EventsFilter

```
/**
 * @(#)com.networkagent.ciscorouter.agent.EventsFilter.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  28 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Filtra mensajes TRAP permitiendo solamente enviar mensajes al
* Administrador si no existe un enlace con alguna de las interfaces
* del ruteador.
*/
import java.util.Vector;
import java.util.HashMap;
import java.util.Iterator;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.framework.util.Filter;
import com.networkagent.ciscorouter.agent.SNMPTrapMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;
import com.networkagent.ciscorouter.mib.MIBConstants;

public class EventsFilter implements Filter
{
   public Object doFilter(Object object) {
      Message message;

      message = (Message)object;
      if (message.getServiceId() == Integer.toHexString(SNMPConstants.SNMP_TRAP)) {
         SNMPTrapMessage  trapMessage;
         Vector       vector;
         HashMap          map;
         Iterator     iterator;
         VariablesBindvarBind;
         String       instance = "";
         trapMessage = (SNMPTrapMessage) message;
         vector    = trapMessage.getVarBindList();
         iterator  = vector.iterator();
         map       = new HashMap();
      while (iterator.hasNext()) {
            varBind    = (VariablesBind) iterator.next();
            instance = varBind.getInstance();
            map.put(varBind.getInstance(), varBind);
         }

         if (instance.startsWith(MIBConstants.lifEntry)) {
            varBind = (VariablesBind) map.get(MIBConstants.locIfLineInt);
            if ( ((Integer)varBind.getValue()).intValue() == 0) {
               return message;
            } else {
               varBind = (VariablesBind) map.get(MIBConstants.locIfLineProt);
               if ( ((Integer)varBind.getValue()).intValue() == 0) {
                  return message;
               } else {
                  return null;
               }
            }
         }
      }
      return message;
   }
}
```

D.2.3 FifoMessageQueue

```
/**
 * @(#)com.networkagent.ciscorouter.agent.FifoMessageQueue.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  24 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Implementa el mecanismo de contención de mensajes FIFO
* (First-In, First-Out).
*/

import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.util.FilterChain;

public class FifoMessageQueue extends MessageQueue {
   public FifoMessageQueue(int maxSize, FilterChain filterChain) {
      super(maxSize, filterChain);
   }

   public Message getNextMessage() {
      return (Message) this.messagesList.remove(0);
   }
}
```

D.2.4 GetService

```
/**
 * @(#)com.networkagent.ciscorouter.agent.GetService.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  02 de Enero de 2003
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Implementación del Servicio GET-REQUEST del protocolo SNMP.
* Permite obtener objetos administrados de la MIB dada una lista
* de variables que discrimina la búsqueda.
*/

import java.util.Vector;
import com.networkagent.framework.agent.Service;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.mib.ManagedObject;
import com.networkagent.framework.mib.MIBManager;
import com.networkagent.ciscorouter.agent.SNMPGetMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;

public class GetService extends Service {
   public GetService(String serviceId, Message message) {
      super(serviceId, message);
   }

   public Message executeCommand() {
      AgentResources   resources;
      MIBManager       mibManager;
      ManagedObject object;
      SNMPGetMessage   getMessage;
      SNMPGetMessage   getResponse;

      resources    = AgentResources.getInstance();
      mibManager   = (MIBManager) resources.getResource(AgentResourcesKeys.MIB_MANAGER);
      getMessage   = (SNMPGetMessage) this.messageIn;
      object    = mibManager.getManagedObject(getMessage.getVarBindList());
      getResponse = new SNMPGetMessage();

      if (object != null) {
         getResponse.setServiceId(Integer.toHexString(SNMPConstants.SNMP_RESPONSE));
         getResponse.setMessageId(this.messageIn.getMessageId());
         getResponse.setCommunityName(getMessage.getCommunityName());
         getResponse.setErrorIndex(0);
         getResponse.setStatusError(0);
         getResponse.setVersion(0);
         getResponse.setVarBindList((Vector) object.getValue());
      } else {
         getResponse.setServiceId(Integer.toHexString(SNMPConstants.SNMP_RESPONSE));
         getResponse.setMessageId(this.messageIn.getMessageId());
         getResponse.setCommunityName(getMessage.getCommunityName());
         getResponse.setErrorIndex(1);
         getResponse.setStatusError(SNMPConstants.SNMP_NOSUCHNAME);
         getResponse.setVersion(0);
         getResponse.setVarBindList((Vector) object.getValue());
      }
      this.messageOut = getResponse;
      return getResponse;
   }
}
```

D.2.5 PoolProcessManager

```
/**
 * @(#)com.networkagent.ciscorouter.agent.PoolProcessManager.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  24 de Noviembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Implementa el administrador de procesos del Agente.
* En este caso se implementa un mecanismo de "pool" de procesos
* para atender los servicios solicitados por el Administrador, o
* en su defecto por reportes de cambio de estado en los Recursos.
*/

import java.util.Vector;
import java.util.HashMap;
import java.util.Iterator;

import com.networkagent.framework.agent.ProcessManager;
import com.networkagent.framework.agent.Process;

public class PoolProcessManager extends ProcessManager {
   private HashMap  busyProcessess;
   private Vector   availableProcessess;

   public PoolProcessManager(int size) {
      super(size);

      Iterator  iterator;

      this.availableProcessess = new Vector();
      this.busyProcessess      = new HashMap();
      iterator             = this.processList.iterator();

      while(iterator.hasNext()) {
         this.availableProcessess.add(iterator.next());
      }
   }

   protected Process getProcess() {
      com.networkagent.framework.agent.Process   process;

      process = (Process) this.availableProcessess.remove(0);
      this.busyProcessess.put(String.valueOf(process.getProcessId()), process);

      return process;
   }

   public void releaseProcess(int processId) {
      Process   process;

      process = (Process) this.busyProcessess.remove(String.valueOf(processId));
      this.availableProcessess.add(process);
   }

   public void suspendProcess(int processId) {

   }
}
```

D.2.6 SNMPConstants

```
/**
 * @(#)com.networkagent.ciscorouter.agent.FifoMessageQueue.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  04 de Enero de 2002
 * @Version:    3.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Constantes SNMP para la interpretación y análisis de mensajes.
*/

public interface SNMPConstants {
   public static final int PACKET_SIZE     = 1024;
   public static final intASN_HEADER          = 0x30;
   public static final intSNMP_INTEGER     = 0x02;
   public static final intSNMP_STRING         = 0x04;
   public static final intSNMP_NULL        = 0x05;
   public static final intSNMP_OBJECTID    = 0x06;
   public static final intSNMP_GET         = 0xa0;
   public static final intSNMP_GETNEXT     = 0xa1;
   public static final intSNMP_RESPONSE    = 0xa2;
   public static final intSNMP_SET         = 0xa3;
   public static final intSNMP_TRAP        = 0xa4;
   public static final intSNMP_VARBINDLIST = 0x30;
   public static final intSNMP_VARBIND     = 0x30;

   public static final intSNMP_NOERROR     = 0x00;
   public static final intSNMP_TOOBIG         = 0x01;
   public static final intSNMP_NOSUCHNAME     = 0x02;
   public static final intSNMP_BADVALUE    = 0x03;
   public static final intSNMP_READONLY    = 0x04;
   public static final intSNMP_GENERR         = 0x05;

   public static final intSNMP_COLDSTART            = 0x00;
   public static final intSNMP_WARMSTART            = 0x01;
   public static final intSNMP_LINKDOWN          = 0x02;
   public static final intSNMP_LINKUP               = 0x03;
   public static final intSNMP_AUTHENTICATIONFAILURE = 0x04;
   public static final intSNMP_EGPNEIGHBORLOSS      = 0x05;
   public static final intSNMP_ENTERPRISESPECIFIC   = 0x06;

   public static final String SNMP_GETREQUEST_TEXT   = "GET-REQUEST";
   public static final String SNMP_TRAP_TEXT         = "TRAP";
   public static final String COMMUNITY_NAME         = "virtualComm0001";
}
```

D.2.7 SNMPGetMessage

```
/**
 * @(#)com.networkagent.ciscorouter.agent.SNMPGetMessage.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */
package com.networkagent.ciscorouter.agent;
/**
* Clase que implementa los mensajes SNMP GET-REQUEST y GET-RESPONSE.
*/
import java.util.Vector;
import com.networkagent.framework.agent.Message;

public class SNMPGetMessage extends Message {
   private int  version;
   private String   communityName;
   private int      statusError;
   private int  errorIndex;
   private Vector   varBindList;

   public SNMPGetMessage() {
      varBindList = new Vector();
   }

   public void setVersion(int version) {
      this.version = version;
   }

   public void setCommunityName(String communityName) {
      this.communityName = communityName;
   }

   public void setStatusError(int statusError) {
      this.statusError = statusError;
   }

   public void setErrorIndex(int errorIndex) {
      this.errorIndex = errorIndex;
   }

   public void setVarBindList(Vector varBindList) {
      this.varBindList = varBindList;
   }

   public int getVersion() {
      return this.version;
   }

   public String getCommunityName() {
      return this.communityName;
   }

   public int getStatusError() {
      return this.statusError;
   }

   public int getErrorIndex() {
      return this.errorIndex;
   }

   public Vector getVarBindList() {
      return this.varBindList;
   }
}
```

D.2.8 SNMPMessageCoder

```
/**
 * @(#)com.networkagent.ciscorouter.agent.SNMPMessageCoder.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  02 de Enero de 2003
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Codificador de mensajes SNMP. En este caso codifica mensajes
* GET-RESPONSE y TRAP.
*/

import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.util.Vector;
import java.util.Iterator;

import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.MessageCoder;
import com.networkagent.framework.agent.VariablesBind;

import com.networkagent.ciscorouter.agent.SNMPGetMessage;
import com.networkagent.ciscorouter.agent.SNMPTrapMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;


public class SNMPMessageCoder implements MessageCoder
{
   public OutputStream code(Message message) {
      ByteArrayOutputStream  byteStream = null;
      DataOutputStream       dataStream;
      int                 byteCounter = 0;
      Iterator            iterator;
      VariablesBind       varBind;

      try {
         byteStream = new ByteArrayOutputStream();
         dataStream = new DataOutputStream(byteStream);

         if (message.getServiceId().equals(Integer.toHexString(SNMPConstants.SNMP_GET))) {
            SNMPGetMessage   getMessage;

            getMessage   = (SNMPGetMessage) message;
            iterator  = getMessage.getVarBindList().iterator();

            dataStream.writeByte(SNMPConstants.ASN_HEADER);
            byteCounter++;
            dataStream.writeInt(0);
            byteCounter += 4;
            dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
            dataStream.writeByte(1);
            dataStream.writeByte(0);
            dataStream.writeByte(getMessage.getCommunityName().length());
            dataStream.writeBytes(getMessage.getCommunityName());
            dataStream.writeByte(SNMPConstants.SNMP_RESPONSE);
            dataStream.writeInt(0);
            dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
            dataStream.writeInt(4);
            dataStream.writeInt(Integer.parseInt(getMessage.getMessageId()));
            dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
```

```
    dataStream.writeInt(4);
    dataStream.writeInt(getMessage.getStatusError());
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeInt(4);
    dataStream.writeInt(getMessage.getErrorIndex());
    dataStream.writeByte(SNMPConstants.SNMP_VARBINDLIST);
    dataStream.writeInt(0);

    while (iterator.hasNext()) {
       varBind = (VariablesBind) iterator.next();
       dataStream.writeByte(SNMPConstants.SNMP_VARBIND);
       dataStream.writeInt(0);
       dataStream.writeByte(SNMPConstants.SNMP_OBJECTID);
       dataStream.writeInt(0);
       dataStream.writeBytes(varBind.getInstance());
       if (varBind.getValue() != null) {
          dataStream.writeByte(varBind.getDataType());
          dataStream.writeInt(varBind.getDataLength());

          switch(varBind.getDataType()) {
             case SNMPConstants.SNMP_INTEGER:
                dataStream.writeInt(((Integer)varBind.getValue()).intValue());
                break;

             case  SNMPConstants.SNMP_STRING:
                dataStream.writeBytes((String) varBind.getValue());
          }

       } else {
          dataStream.writeByte(SNMPConstants.SNMP_NULL);
          dataStream.writeByte(0);
       }
    }
} else if (message.getServiceId().equals( Integer.toHexString
(SNMPConstants.SNMP_TRAP))) {
    SNMPTrapMessage  trapMessage;

    trapMessage = (SNMPTrapMessage) message;
    iterator  = trapMessage.getVarBindList().iterator();

    dataStream.writeByte(SNMPConstants.ASN_HEADER);
    byteCounter++;
    dataStream.writeInt(0);
    byteCounter += 4;
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeByte(1);
    dataStream.writeByte(0);
    dataStream.writeByte(SNMPConstants.COMMUNITY_NAME.length());
    dataStream.writeBytes(SNMPConstants.COMMUNITY_NAME);
    dataStream.writeByte(SNMPConstants.SNMP_TRAP);
    dataStream.writeInt(0);
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeInt(4);
    dataStream.writeInt(Integer.parseInt(trapMessage.getMessageId()));
    dataStream.writeByte(SNMPConstants.SNMP_OBJECTID);
    dataStream.writeInt(0);
    dataStream.writeBytes(trapMessage.getEnterprise());
    dataStream.writeByte(SNMPConstants.SNMP_STRING);
    dataStream.writeInt(0);
    dataStream.writeBytes(trapMessage.getAgentAddress());
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeInt(4);
    dataStream.writeInt(trapMessage.getGenericTrap());
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeInt(4);
    dataStream.writeInt(trapMessage.getSpecificTrap());
    dataStream.writeByte(SNMPConstants.SNMP_INTEGER);
    dataStream.writeInt(8);
    dataStream.writeLong(trapMessage.getTimeStamp());
```

```
            while (iterator.hasNext()) {
               varBind = (VariablesBind) iterator.next();
               dataStream.writeByte(SNMPConstants.SNMP_VARBIND);
               dataStream.writeInt(0);
               dataStream.writeByte(SNMPConstants.SNMP_OBJECTID);
               dataStream.writeInt(0);
               dataStream.writeBytes(varBind.getInstance());
               if (varBind.getValue() != null) {
                  dataStream.writeByte(varBind.getDataType());
                  dataStream.writeInt(varBind.getDataLength());

                  switch(varBind.getDataType()) {
                     case SNMPConstants.SNMP_INTEGER:
                        dataStream.writeInt(((Integer)varBind.getValue()).intValue());
                        break;

                     case  SNMPConstants.SNMP_STRING:
                        dataStream.writeBytes((String) varBind.getValue());
                     }

               } else {
                  dataStream.writeByte(SNMPConstants.SNMP_NULL);
                  dataStream.writeByte(0);
               }
            }
         }
      } catch (IOException ioexc) {
         System.out.println ("Error en la generación de mensajes. Excepción: " +
         ioexc.toString());
      }

      return byteStream;
   }
}
```

D.2.9 SNMPMessageDecoder

```
/**
 * @(#)com.networkagent.ciscorouter.agent.SNMPMessageDecoder.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  26 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Decodificador de mensajes SNMP. En este caso convierte un flujo
* de datos de entrada en mensajes GET-REQUEST.
*/

import java.io.DataInputStream;
import java.io.InputStream;
import java.io.IOException;

import com.networkagent.framework.agent.MessageDecoder;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.agent.VariablesBind;

import com.networkagent.ciscorouter.agent.SNMPConstants;
import com.networkagent.ciscorouter.agent.SNMPObject;
import com.networkagent.ciscorouter.agent.SNMPGetMessage;


public class SNMPMessageDecoder implements MessageDecoder {

   public Message decode(InputStream stream) {
      DataInputStream dataStream;
      byte         header;
      short        pduLength;
      int          varBindListLength;
      int          varBindLength;
      int          varBindCounter;
      VariablesBind varBind;

      Message         message;

      SNMPObject      version;
      SNMPObject      communityName;
      SNMPObject      data;
      SNMPObject      requestType;
      SNMPGetMessage  getMessage;

      dataStream      = new DataInputStream(stream);
      varBindCounter  = 0;

      try {
         header    = dataStream.readByte();

         if (header == SNMPConstants.ASN_HEADER) {
            pduLength       = dataStream.readShort();
            version         = this.getObject(dataStream);
            communityName= this.getObject(dataStream);
            requestType     = this.getObject(dataStream);

            switch(requestType.getType()) {
               case SNMPConstants.SNMP_GET:
                  getMessage = new SNMPGetMessage();

                  getMessage.setServiceId(Integer.toHexString(requestType.getType()));
```

```
                    System.out.println       ("Servicio       Solicitado:       "       +
Integer.toHexString(requestType.getType()));
                    getMessage.setCommunityName((String) communityName.getObject());
                    getMessage.setVersion((int)((Byte)version.getObject()).byteValue());

                    // Identificador de Mensaje.
                    data = this.getObject(dataStream);
                    getMessage.setMessageId(((Integer)data.getObject()).toString());

                    // Status de Error.
                    data = this.getObject(dataStream);
                    getMessage.setStatusError((int)((Byte)data.getObject()).byteValue());

                    // Indice de variable que genera error.
                    data = this.getObject(dataStream);
                    getMessage.setErrorIndex((int)((Byte)data.getObject()).byteValue());

                    // Comienza Var Bind List
                    data = this.getObject(dataStream);
                    if (data.getType() != SNMPConstants.SNMP_VARBINDLIST) {
                       System.out.println ("No es correcto el formato enviado ya queno
cuenta con el identificador de VarBind List");
                       return null;
                    } else {
                       varBindListLength = data.getLength();
                    }

                    while(varBindCounter < varBindListLength) {
                       // Se obtiene el Var Bind.
                       data = this.getObject(dataStream);
                       if (data.getType() != SNMPConstants.SNMP_VARBIND) {
                          System.out.println ("No es correcto el formato enviado ya queno
cuenta con el identificador de VarBind");
                          return null;
                       }
                       varBindCounter += data.getLength();

                       // Se obtiene el Identificador de Objeto
                       data   = this.getObject(dataStream);
                       varBind   = new VariablesBind();

                       varBind.setInstance((String) data.getObject());

                       // Se obtiene el Objeto.
                       data   = this.getObject(dataStream);
                       varBind.setDataLength(data.getLength());
                       varBind.setDataType(data.getType());
                       varBind.setValue(data.getObject());
                       getMessage.getVarBindList().add(varBind);
                    }

                    message = getMessage;
                    break;
                }
             }
          } catch(IOException ioexc) {
             System.out.println (ioexc.toString());
          }

          return null;
       }

       private SNMPObject getObject(DataInputStream dataStream) throws IOException {
          int       dataType;
          int       dataLength;
          byte []      dataValue;
          SNMPObject   object;

          dataType  = dataStream.readUnsignedByte();
          dataLength   = dataStream.readUnsignedByte();
```

```
      object    = new SNMPObject();

      object.setType(dataType);
      object.setLength(dataLength);

      switch(dataType) {
         case SNMPConstants.SNMP_INTEGER:
            switch(dataLength) {
               case 0x01:
                  object.setObject(new Byte(dataStream.readByte()));
                  break;
               case 0x02:
                  object.setObject(new Short(dataStream.readShort()));
                  break;
               case 0x04:
                  object.setObject(new Integer(dataStream.readInt()));
                  break;
            }
            break;

         case SNMPConstants.SNMP_STRING:
            dataValue    = new byte[dataLength];
            dataStream.read(dataValue);
            object.setObject(new String(dataValue));

         case SNMPConstants.SNMP_OBJECTID:
            String    data;

            dataValue    = new byte[dataLength];
            data      = "";
            dataStream.read(dataValue);

            for(int i=0; i < dataValue.length-1; i++) {
               data += Byte.toString(dataValue[i]) + ".";
            }

            data += Byte.toString(dataValue[dataValue.length-1]);
            object.setObject(data);
            break;

         default:
            object.setObject(null);
            break;
      }

      return object;
   }
}
```

D.2.10 SNMPObject

```
/**
 * @(#)com.networkagent.ciscorouter.agent.SNMPObject.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  26 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Clase que contiene información referente a una variable SNMP, la
* cual contiene el tipo de dato, la logitud y el valor de la variable.
*/

public class SNMPObject {
   private int  type;
   private int  length;
   private Object   object;

   public SNMPObject() {
      super();
   }

   public SNMPObject(int type, int lenght, Object object) {
      this.type    = type;
      this.length  = length;
      this.object  = object;
   }

   public void setType(int type) {
      this.type = type;
   }

   public void setLength(int length) {
      this.length = length;
   }

   public void setObject(Object object) {
      this.object = object;
   }

   public int getType() {
      return this.type;
   }

   public int getLength() {
      return this.length;
   }

   public Object getObject() {
      return this.object;
   }
}
```

D.2.11 SNMPTrapMessage

```
/**
 * @(#)com.networkagent.ciscorouter.agent.SNMPTrapMessage.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  15 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Clase que implementa mensajes TRAP de SNMP.
*/

import java.util.Vector;
import com.networkagent.framework.agent.ReportMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;

public class SNMPTrapMessage extends ReportMessage
{
   private String   enterprise;
   private String   agentAddress;
   private int  genericTrap;
   private int  specificTrap;
   private long     timeStamp;
   private Vector   varBindList;

   public SNMPTrapMessage() {
      super();
      this.setServiceId(Integer.toHexString(SNMPConstants.SNMP_TRAP));
   }

   public String getEnterprise() {
      return this.enterprise;
   }

   public String getAgentAddress() {
      return this.agentAddress;
   }

   public int getGenericTrap() {
      return this.genericTrap;
   }

   public int getSpecificTrap() {
      return this.specificTrap;
   }

   public long getTimeStamp() {
      return this.timeStamp;
   }

   public Vector getVarBindList() {
      return this.varBindList;
   }

   public void setEnterprise(String enterprise) {
      this.enterprise = enterprise;
   }

   public void setAgentAddress(String agentAddress) {
      this.agentAddress = agentAddress;
   }

   public void setGenericTrap(int genericTrap) {
```

```
        this.genericTrap = genericTrap;
    }

    public void setSpecificTrap(int specificTrap) {
        this.specificTrap = specificTrap;
    }

    public void setTimeStamp(long timeStamp) {
        this.timeStamp = timeStamp;
    }

    public void setVarBindList(Vector varBindList) {
        this.varBindList = varBindList;
    }
}
```

D.2.12 TrapService

```
/**
 * @(#)com.networkagent.ciscorouter.agent.GetService.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  02 de Enero de 2003
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Implementación del Servicio TRAP del protocolo SNMP.
* Permite reportar cambios de estado en los Recursos del
* elemento de red.
*/

import com.networkagent.framework.agent.Service;
import com.networkagent.framework.agent.Message;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.ciscorouter.agent.SNMPTrapMessage;

public class TrapService extends Service
{

   public TrapService(String serviceId, Message message) {
      super(serviceId, message);
   }

   public Message executeCommand() {
      SNMPTrapMessage trapMessage;
      AgentResources  agentResources;

      agentResources  = AgentResources.getInstance();
      trapMessage  = (SNMPTrapMessage) this.messageIn;
      trapMessage.setAgentAddress((String) agentResources.getResource(
                                  AgentResourcesKeys.AGENT_IP));

      this.messageOut  = trapMessage;
      return trapMessage;
   }
}
```

D.2.13 UDPMessageReceptor

```
/**
 * @(#)com.networkagent.ciscorouter.agent.UDPMessageReceptor.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  20 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */
package com.networkagent.ciscorouter.agent;

/**
* Receptor de mensajes SNMP, en este caso implementa el mecanismo de
* datagramas para la recepción de mensajes.
*/

import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.SocketException;
import java.io.InputStream;
import java.io.IOException;
import java.io.ByteArrayInputStream;
import com.networkagent.framework.agent.MessageReceptor;
import com.networkagent.framework.agent.MessageDecoder;

public class UDPMessageReceptor extends MessageReceptor {
   private   DatagramSocket  udpServer;
   private int            packetLength;

   public UDPMessageReceptor(int port, MessageDecoder decoder) {
      super(port, decoder);
      packetLength = 1024;
   }

   protected void initReceptor() {
      try {
         this.udpServer = new DatagramSocket(this.port);
      } catch(SocketException sockexc) {
         System.out.println ("Error al intentar iniciar el receptor de Mensajes..." +
                             sockexc.toString());
         System.exit(0);
      }
   }

   protected InputStream receiveData() {
      DatagramPacket         packet;
      ByteArrayInputStream   stream;
      byte[]              buffer;

      buffer = new byte[packetLength];
      packet = new DatagramPacket(buffer, buffer.length);

      try {
         this.udpServer.receive(packet);
      } catch (IOException ioexc)   {
         System.out.println ("Error al intenter recibir un paquete..." + ioexc.toString());
         return null;
      }

      stream = new ByteArrayInputStream(packet.getData());
      return stream;
   }

   protected void destroyReceptor() {
      this.udpServer.close();
   }
}
```

D.2.14 UDPMessageReceptor

```
/**
 * @(#)com.networkagent.ciscorouter.agent.UDPMessageTransmisor.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  20 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.agent;

/**
* Transmisor de mensajes SNMP, en este caso implementa el mecanismo de
* datagramas para la emisión de mensajes.
*/

import java.io.ByteArrayOutputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.SocketException;

import com.networkagent.framework.agent.MessageTransmisor;
import com.networkagent.framework.agent.AdministratorDirVO;

public class UDPMessageTransmisor extends MessageTransmisor {

   protected void sendData(OutputStream stream, AdministratorDirVO administratorDir) {
      ByteArrayOutputStream byteStream;
      byte[]            buffer;
      DatagramPacket        packet;
      InetAddress           address;
      InetSocketAddress     inetAddress;
      DatagramSocket        socket;

      inetAddress  =            new            InetSocketAddress(administratorDir.getIP(),
administratorDir.getPort());
      address      = inetAddress.getAddress();
      byteStream   = (ByteArrayOutputStream) stream;
      buffer    = byteStream.toByteArray();
      packet    =       new       DatagramPacket(buffer,       buffer.length,       address,
administratorDir.getPort());

      try {
         socket    = new DatagramSocket(administratorDir.getPort());
         socket.send(packet);
      } catch (SocketException sockexc) {
         System.out.println  ("Error  al  intentar  crear  un  socket  datagrama..."  +
sockexc.toString());
         return;
      } catch (IOException ioexc)  {
         System.out.println   ("Error   al   intentar   enviar   un   datagrama..."   +
ioexc.toString());
         return;
      }
   }
}
```

D.3 Implementación de la Capa "MIB"

D.3.1 LIfEntry

```
/**
 * @(#)com.networkagent.ciscorouter.mib.LifEntry.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  29 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.mib;

/**
* Clase que representa a un elemento de la tabla LIfTable de SNMP,
* el cual permite contener información del estado de interaces del
* ruteador Cisco 2501.
*/

import java.util.HashMap;

import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.framework.mib.ManagedObject;
import com.networkagent.ciscorouter.mib.MIBConstants;
import com.networkagent.ciscorouter.agent.SNMPConstants;

public class LIfEntry extends ManagedObject {
   private int  ifIndex;
   private String   locIfDescr;
   private String   locIfHardType;
   private int  locIfLineProt;
   private int  locIfLineInt;
   private int  locIfCollisions;
   private int  locIfKeep;
   private int  locIfInAbort;
   private int  locIfResets;
   private int  locIfInRunts;
   private int  locIfInGiants;
   private int  locIfInCRC;
   private int  locIfInFrame;
   private int  locIfInOverrun;
   private int  locIfInIgnored;
   private int  locIfipInPkts;
   private int  locIfipOutPkts;
   private int  locIfipInOctects;
   private int  locIfipOutOctects;
   private HashMap  values;

    public LIfEntry(String id, String instance, boolean read, boolean write) {
      super(id, instance, read, write);

      VariablesBind varBind;

      this.values = new HashMap();

      varBind = new VariablesBind();
      varBind.setInstance(MIBConstants.ifIndex);
      varBind.setDataLength(0x04);
      varBind.setDataType(SNMPConstants.SNMP_INTEGER);
      this.values.put(varBind.getInstance(), varBind);

      varBind = new VariablesBind();
      varBind.setInstance(MIBConstants.locIfDescr);
      varBind.setDataLength(0x00);
      varBind.setDataType(SNMPConstants.SNMP_STRING);
```

```
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfHardType);
varBind.setDataLength(0x00);
varBind.setDataType(SNMPConstants.SNMP_STRING);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfLineProt);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfLineInt);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfCollisions);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfKeep);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInAbort);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfResets);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInRunts);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInGiants);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInCRC);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInFrame);
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
this.values.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setInstance(MIBConstants.locIfInOverrun);
```

```
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.locIfInIgnored);
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.locIfipInPkts);
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.locIfipOutPkts);
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.locIfipInOctects);
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.locIfipOutOctects);
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        this.values.put(varBind.getInstance(), varBind);
    }

    public int getIfIndex() {
        return this.ifIndex;
    }

    public String getLocIfDescr() {
        return this.locIfDescr;
    }

    public String getLocIfHardType() {
        return this.locIfHardType;
    }

    public int getLocIfLineProt() {
        return this.locIfLineProt;
    }

    public int getLocIfLineInt() {
        return this.locIfLineInt;
    }

    public int getLocIfCollisions() {
        return this.locIfCollisions;
    }

    public int getLocIfKeep() {
        return this.locIfKeep;
    }

    public int getLocIfInAbort() {
        return this.locIfInAbort;
    }

    public int getLocIfInRunts() {
        return this.locIfInRunts;
    }
```

```
public int getLocIfInGiants() {
   return this.locIfInGiants;
}

public int getLocIfInCRC() {
   return this.locIfInCRC;
}

public int getLocIfInFrame() {
   return this.locIfInFrame;
}

public int getLocIfInOverrun() {
   return this.locIfInOverrun;
}

public int getLocIfInIgnored() {
   return this.locIfInIgnored;
}

public int getLocIfipInPkts() {
   return this.locIfipInPkts;
}

public int getLocIfipOutPkts() {
   return this.locIfipOutPkts;
}

public int getLocIfipInOctects() {
   return this.locIfipInOctects;
}

public int getLocIfipOutOctects() {
   return this.locIfipOutOctects;
}

public Object getValue() {
   VariablesBind varBind;

   varBind = (VariablesBind) this.values.get(MIBConstants.ifIndex);
   varBind.setValue(new Integer(this.ifIndex));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfDescr);
   varBind.setValue(this.locIfDescr);

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfHardType);
   varBind.setValue(this.locIfHardType);

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfLineProt);
   varBind.setValue(new Integer(this.locIfLineProt));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfLineInt);
   varBind.setValue(new Integer(this.locIfLineInt));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfCollisions);
   varBind.setValue(new Integer(this.locIfCollisions));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfKeep);
   varBind.setValue(new Integer(this.locIfKeep));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfInAbort);
   varBind.setValue(new Integer(this.locIfInAbort));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfResets);
   varBind.setValue(new Integer(this.locIfResets));

   varBind = (VariablesBind) this.values.get(MIBConstants.locIfInRunts);
   varBind.setValue(new Integer(this.locIfInRunts));
```

```
        varBind = (VariablesBind) this.values.get(MIBConstants.locIfInGiants);
        varBind.setValue(new Integer(this.locIfInGiants));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfInCRC);
        varBind.setValue(new Integer(this.locIfInCRC));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfInFrame);
        varBind.setValue(new Integer(this.locIfInFrame));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfInOverrun);
        varBind.setValue(new Integer(this.locIfInOverrun));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfInIgnored);
        varBind.setValue(new Integer(this.locIfInIgnored));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfipInPkts);
        varBind.setValue(new Integer(this.locIfipInPkts));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfipOutPkts);
        varBind.setValue(new Integer(this.locIfipOutPkts));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfipInOctects);
        varBind.setValue(new Integer(this.locIfipInOctects));

        varBind = (VariablesBind) this.values.get(MIBConstants.locIfipOutOctects);
        varBind.setValue(new Integer(this.locIfipOutOctects));

        return this.values;
    }

    public boolean getState() {
        return false;
    }

    public void setIfIndex(int ifIndex) {
        this.ifIndex = ifIndex;
    }


    public void setLocIfDescr(String locIfDescr) {
        this.locIfDescr = locIfDescr;
    }

    public void setLocIfHardType(String locIfHardType) {
        this.locIfHardType = locIfHardType;
    }

    public void setLocIfLineProt(int locIfLineProt) {
        this.locIfLineProt = locIfLineProt;
    }

    public void setLocIfLineInt(int locIfLineInt) {
        this.locIfLineInt = locIfLineInt;
    }

    public void setLocIfCollisions(int locIfCollisions) {
        this.locIfCollisions = locIfCollisions;
    }

    public void setLocIfKeep(int locIfKeep) {
        this.locIfKeep = locIfKeep;
    }

    public void setLocIfInAbort(int locIfInAbort) {
        this.locIfInAbort = locIfInAbort;
    }

    public void setLocIfResets(int locIfResets){
        this.locIfResets = locIfResets;
    }
```

```
    public void setLocIfInRunts(int locIfInRunts) {
        this.locIfInRunts = locIfInRunts;
    }

    public void setLocIfInGiants(int locIfInGiants) {
        this.locIfInGiants = locIfInGiants;
    }

    public void setLocIfInCRC(int locIfInCRC) {
        this.locIfInCRC = locIfInCRC;
    }

    public void setLocIfInFrame(int locIfInFrame) {
        this.locIfInFrame = locIfInFrame;
    }

    public void setLocIfInOverrun(int locIfInOverrun) {
        this.locIfInOverrun = locIfInOverrun;
    }

    public void setLocIfInIgnored(int locIfInIgnored) {
        this.locIfInIgnored = locIfInIgnored;
    }

    public void setLocIfipInPkts(int locIfipInPkts) {
        this.locIfipInPkts = locIfipInPkts;
    }

    public void setLocIfipOutPkts(int locIfipOutPkts) {
        this.locIfipOutPkts = locIfipOutPkts;
    }

    public void setLocIfipInOctects(int locIfipInOctects) {
        this.locIfipInOctects = locIfipInOctects;
    }

    public void setLocIfipOutOctects(int locIfipOutOctects) {
        this.locIfipOutOctects = locIfipOutOctects;
    }

    public void setValue(Object value) {
        this.values = (HashMap) value;
    }
}
```

D.3.2 MIBConstants

```
/**
 * @(#)com.networkagent.ciscorouter.mib.MIBConstants.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  03 de Enero de 2003
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.mib;

/**
* Constantes utilizadas en la MIB.
*/

public interface MIBConstants {
   public static final String cisco             = "1.3.6.5";
   public static final String mib            = cisco + ".1";
   public static final String local          = mib + ".4";
   public static final String linterfaces       = local + ".2";
   public static final String lifTable          = linterfaces + ".1";
   public static final String lifEntry          = lifTable + ".1";
   public static final String ifIndex           = lifEntry + ".0";
   public static final String locIfHardType     = lifEntry + ".1";
   public static final String locIfLineProt     = lifEntry + ".2";
   public static final String locIfLineInt      = lifEntry + ".3";
   public static final String locIfInRunts      = lifEntry + ".10";
   public static final String locIfInGiants     = lifEntry + ".11";
   public static final String locIfInCRC        = lifEntry + ".12";
   public static final String locIfInFrame      = lifEntry + ".13";
   public static final String locIfInOverrun    = lifEntry + ".14";
   public static final String locIfInIgnored    = lifEntry + ".15";
   public static final String locIfInAbort      = lifEntry + ".16";
   public static final String locIfResets       = lifEntry + ".17";
   public static final String locIfKeep      = lifEntry + ".19";
   public static final String locIfCollisions   = lifEntry + ".25";
   public static final String locIfDescr        = lifEntry + ".28";
   public static final String locIfipInPkts     = lifEntry + ".42";
   public static final String locIfipOutPkts    = lifEntry + ".43";
   public static final String locIfipInOctects      = lifEntry + ".44";
   public static final String locIfipOutOctects  = lifEntry + ".45";
}
```

D.3.3 MIBDBContainer

```
/**
 * @(#)com.networkagent.ciscorouter.mib.MIBDBContainer.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  22 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.mib;

/**
* Clase que implementa el contenedor de Objetos Administrados de la
* MIB. En esta caso se implementa un contenedor de Base de Datos, el
* cual utiliza el puente JDBC-ODBC para conectarse con la BD en Access.
*/

import java.util.Vector;
import java.util.Iterator;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

import com.networkagent.framework.mib.MIBContainer;
import com.networkagent.framework.mib.ManagedObject;
import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.ciscorouter.mib.MIBConstants;
import com.networkagent.ciscorouter.mib.LIfEntry;

public class MIBDBContainer implements MIBContainer {
   private String selLIfEntry    = "SELECT ifIndex, locIfDesc, locIfHardTypw, locIfLineProt,
" +
                               "      locIfLineInt, locIfCollisions, locIfKeep, locIfInAbort, "
+
                               "       locIfResets, locIfInRunts, locIfInGiants, locIfInCRC, "
+
                               "       locIfInFrame, locIfInOverrun, locIfInIgnored,
locIfipInPkts, " +
                               "      locIfipOutPkts, locIfipInOctects, locIfipOutOctects " +
                               "FROM     LIfTable ";
   private String delLIfTable  = "DELETE FROM LIfTable";
   private String updLIfEntry = "UPDATE LIfTable ";
   private String insLIfEntry = "INSERT INTO LIfTable ";
   private String connString  = "jdbc:odbc:mib";

   public void initialize() {
      Connection   conn = null;
      Statement stmt;

      try {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      } catch (ClassNotFoundException cnfexc) {
         System.out.println("ClassNotFoundException: "+ cnfexc.toString());
      }

      // Elimina los registros de la tablas LIfTable para despues llenarlas
      // con registros de la AVH.
      try{
         conn  = DriverManager.getConnection(this.connString,"","");
         stmt  = conn.createStatement();
         stmt.executeUpdate(this.delLIfTable);
      } catch (SQLException sqlexc) {
         System.err.println("SQLException: "+ sqlexc.toString());
      }
```

```
   }

   public ManagedObject getManagedObject(Vector varBindList) {
      VariablesBind   varBind;
      String       select;
      String       instance;
      String       and = "";
      Connection      conn = null;
      Statement    stmt;
      ResultSet    rs;
      Iterator     iterator;

      if (varBindList != null) {
         varBind  = (VariablesBind) varBindList.firstElement();
         iterator = varBindList.iterator();

         if (varBind.getInstance().startsWith(MIBConstants.lifEntry)) {
            select  = this.selLIfEntry;
            select += "WHERE (";

            while(iterator.hasNext()) {
               varBind   = (VariablesBind) iterator.next();
               instance  = varBind.getInstance();

               if (instance.startsWith(MIBConstants.ifIndex)) {
                  select += and + "(ifIndex = " + ((Integer)
varBind.getValue()).intValue() + ")";
                  and    = " AND ";
               } else if (instance.startsWith(MIBConstants.locIfDescr)) {
                  select += and + "(locIfDescr = '" + (String)varBind.getValue() + "')";
                  and    = " AND ";
               }
            }

            select += ") ";

            try {
               conn = DriverManager.getConnection(this.connString,"","");
               stmt = conn.createStatement();
               rs   = stmt.executeQuery(select);

               if (rs.next()) {
                  LIfEntry lifEntry = new LIfEntry(varBind.getInstance(),
String.valueOf(varBind.getInstance().charAt(varBind.getInstance().length()-1)), true, true);

                  lifEntry.setIfIndex(rs.getInt(1));
                  lifEntry.setLocIfDescr(rs.getString(2));
                  lifEntry.setLocIfHardType(rs.getString(3));
                  lifEntry.setLocIfLineProt(rs.getInt(4));
                  lifEntry.setLocIfLineInt(rs.getInt(5));
                  lifEntry.setLocIfCollisions(rs.getInt(6));
                  lifEntry.setLocIfKeep(rs.getInt(7));
                  lifEntry.setLocIfInAbort(rs.getInt(8));
                  lifEntry.setLocIfResets(rs.getInt(9));
                  lifEntry.setLocIfInRunts(rs.getInt(10));
                  lifEntry.setLocIfInGiants(rs.getInt(11));
                  lifEntry.setLocIfInCRC(rs.getInt(12));
                  lifEntry.setLocIfInFrame(rs.getInt(13));
                  lifEntry.setLocIfInOverrun(rs.getInt(14));
                  lifEntry.setLocIfInIgnored(rs.getInt(15));
                  lifEntry.setLocIfipInPkts(rs.getInt(16));
                  lifEntry.setLocIfipOutPkts(rs.getInt(17));
                  lifEntry.setLocIfipInOctects(rs.getInt(18));
                  lifEntry.setLocIfipOutOctects(rs.getInt(19));

                  return lifEntry;
               }
            } catch (SQLException sqlexc) {
               System.out.println ("Error al intentar obtener LIFEntr. Excepción: " +
sqlexc.toString());
```

```
                return null;
             } finally {
                try {
                   if (conn != null) {
                      conn.close();
                   }
                } catch(SQLException sqlexc) {
                   System.out.println ("Error al intentar cerrar la conexion. Excepción: "
+ sqlexc.toString());
                }
             }
          }
       }

       return null;
   }

   public ManagedObject setManagedObject(Vector varBindList) {
       VariablesBind   varBind;
       String       update;
       String       where;
       String       instance;
       String       and   = "";
       String       comma = "";
       Connection      conn  = null;
       Statement    stmt;
       ResultSet    rs;
       Iterator     iterator;
       int          index = 0;

       if (varBindList != null) {
          varBind  = (VariablesBind) varBindList.firstElement();
          iterator = varBindList.iterator();

          if (varBind.getInstance().startsWith(MIBConstants.lifEntry)) {
             update  = this.updLIfEntry;
             update += "SET ";

             while(iterator.hasNext()) {
                varBind   = (VariablesBind) iterator.next();
                instance  = varBind.getInstance();

                if (instance.startsWith(MIBConstants.ifIndex)) {
                   index  = ((Integer) varBind.getValue()).intValue();
                   where  = " WHERE (ifIndex = " + index + ") ";
                } else if (instance.startsWith(MIBConstants.locIfDescr)) {
                   update += comma + "locIfDescr = '" + (String) varBind.getValue() + "' ";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfHardType)) {
                   update += comma + "locIfHardType = '" + (String) varBind.getValue() + "'
";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfLineProt)) {
                   update += comma + "locIfLineProt = " + ((Integer)
varBind.getValue()).intValue() + " ";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfLineInt)) {
                   update += comma + "locIfLineInt = " + ((Integer)
varBind.getValue()).intValue() + " ";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfCollisions)) {
                   update += comma + "locIfCollisions = " + ((Integer)
varBind.getValue()).intValue() + " ";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfKeep)) {
                   update += comma + "locIfKeep = " + ((Integer)
varBind.getValue()).intValue() + " ";
                   comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInAbort)) {
```

```
                    update += comma + "locIfInAbort = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfResets)) {
                    update += comma + "locIfResets = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInRunts)) {
                    update += comma + "locIfInRunts = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInGiants)) {
                    update += comma + "locIfInGiants = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInCRC)) {
                    update += comma + "locIfInCRC = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInFrame)) {
                    update += comma + "locIfInFrame = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInOverrun)) {
                    update += comma + "locIfInOverrun = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfInIgnored)) {
                    update += comma + "locIfInIgnored = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfipInPkts)) {
                    update += comma + "locIfipInPkts = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfipOutPkts)) {
                    update += comma + "locIfipOutPkts = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfipInOctects)) {
                    update += comma + "locIfipInOctects = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                } else if (instance.startsWith(MIBConstants.locIfipOutOctects)) {
                    update += comma + "locIfipOutOctects = " + ((Integer)
varBind.getValue()).intValue() + " ";
                    comma   = ",";
                }
            }

            try {
                Vector          vector;

                conn = DriverManager.getConnection(this.connString,"","");
                stmt = conn.createStatement();
                stmt.executeUpdate(update);

                vector = new Vector();
                varBind = new VariablesBind();
                varBind.setInstance(MIBConstants.ifIndex);
                varBind.setValue(new Integer(index));
                vector.add(vector);

                return this.getManagedObject(vector);
            } catch (SQLException sqlexc) {
                System.out.println ("Error al intentar actualizar LIFEntr. Excepción: " +
sqlexc.toString());
                return null;
            } finally {
                try {
```

```
                  if (conn != null) {
                     conn.close();
                  }
               } catch(SQLException sqlexc) {
                  System.out.println ("Error al intentar cerrar la conexion. Excepción: "
+ sqlexc.toString());
               }
            }
         }
      }

      return null;
   }

   public ManagedObject insertManagedObject(Vector varBindList) {
      VariablesBind   varBind;
      String       insert;
      String       columns;
      String       values;
      String       instance;
      String       comma = "";
      Connection      conn  = null;
      Statement    stmt;
      ResultSet    rs;
      Iterator     iterator;
      int          index = 0;

      if (varBindList != null) {
         varBind  = (VariablesBind) varBindList.firstElement();
         iterator = varBindList.iterator();

         if (varBind.getInstance().startsWith(MIBConstants.lifEntry)) {
            insert  = this.insLIfEntry;
            columns = "(";
            values  = " VALUES(";

            while(iterator.hasNext()) {
               varBind   = (VariablesBind) iterator.next();
               instance  = varBind.getInstance();

               if (instance.startsWith(MIBConstants.ifIndex)) {
                  index   = ((Integer) varBind.getValue()).intValue();
                  columns += comma + "ifIndex";
                  values  += comma + index;
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfDescr)) {
                  columns += comma + "locIfDescr";
                  values += comma + (String) varBind.getValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfHardType)) {
                  columns += comma + "locIfHardType";
                  values += comma + (String) varBind.getValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfLineProt)) {
                  columns += comma + "locIfLineProt";
                  values += comma + ((Integer) varBind.getValue()).intValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfLineInt)) {
                  columns += comma + "locIfLineInt";
                  values += comma + ((Integer) varBind.getValue()).intValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfCollisions)) {
                  columns += comma + "locIfCollisions";
                  values += comma + ((Integer) varBind.getValue()).intValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfKeep)) {
                  columns += comma + "locIfKeep";
                  values += comma + ((Integer) varBind.getValue()).intValue();
                  comma   = ",";
               } else if (instance.startsWith(MIBConstants.locIfInAbort)) {
```

```
            columns += comma + "locIfInAbort";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfResets)) {
            columns += comma + "locIfResets";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInRunts)) {
            columns += comma + "locIfInRunts";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInGiants)) {
            columns += comma + "locIfInGiants";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInCRC)) {
            columns += comma + "locIfInCRC";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInFrame)) {
            columns += comma + "locIfInFrame";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInOverrun)) {
            columns += comma + "locIfInOverrun";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfInIgnored)) {
            columns += comma + "locIfInIgnored";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfipInPkts)) {
            columns += comma + "locIfipInPkts";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfipOutPkts)) {
            columns += comma + "locIfipOutPkts";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfipInOctects)) {
            columns += comma + "locIfipInOctects";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        } else if (instance.startsWith(MIBConstants.locIfipOutOctects)) {
            columns += comma + "locIfipOutOctects";
            values += comma + ((Integer) varBind.getValue()).intValue();
            comma   = ",";
        }
    }

    columns += ")";
    values += ")";

    insert += columns + values;

    try {
        Vector          vector;

        conn = DriverManager.getConnection(this.connString,"","");
        stmt = conn.createStatement();
        stmt.executeUpdate(insert);

        vector = new Vector();
        varBind = new VariablesBind();
        varBind.setInstance(MIBConstants.ifIndex);
        varBind.setValue(new Integer(index));
        vector.add(vector);

        return this.getManagedObject(vector);
    } catch (SQLException sqlexc) {
```

```
                System.out.println ("Error al intentar insertar LIFEntr. Excepción: " +
sqlexc.toString());
                return null;
             } finally {
                try {
                   if (conn != null) {
                      conn.close();
                   }
                } catch(SQLException sqlexc) {
                   System.out.println ("Error al intentar cerrar la conexion. Excepción: "
+ sqlexc.toString());
                }
             }
          }
       }

       return null;
   }

   public ManagedObject deleteManagedObject(Vector varBindList) {
       return null;
   }
}
```

D.4 Implementación de la Capa "AVH"

D.4.1 AVHConstants

```
/**
 * @(#)com.networkagent.ciscorouter.avh.AVHConstants.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  20 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Constantes utilizadas en la AVH.
*/

public interface AVHConstants {
   public static final String    TELNET_MANAGER   = "TelnetManager";
   public static final int    CHANGE_VALUES    = 1;
   public static final int    LINE_PROT_DOWN   = 2;
   public static final int    LINE_INT_DOWN    = 3;
   public static final int    FATAL_LINE_ERROR = 4;
}
```

D.4.2 RandomResourcesIterator

```
/**
 * @(#)com.networkagent.ciscorouter.avh.RandomResourcesIterator.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  22 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que implementa el mecanismo de iteración de Recursos del
* elemento de red, el cual es utilizado por el AVHManager para monitorear
* dichos Recursos.
*/

import java.util.Vector;
import java.util.Random;
import java.util.Date;

import com.networkagent.framework.avh.ResourcesIterator;
import com.networkagent.framework.avh.Resource;

public class RandomResourcesIterator extends ResourcesIterator {
   private Random random;

   public RandomResourcesIterator(Vector resourcesList) {
      super(resourcesList);

      random = new Random(new Date().getTime());
   }

   public Resource getNext() {
      Resource resource;
      int     number;

      number  = this.random.nextInt(this.resourcesList.size());
      resource = (Resource) this.resourcesList.get(number);

      return resource;
   }
}
```

D.4.3 RouterInterface

```
/**
 * @(#)com.networkagent.ciscorouter.avh.RouterInterface.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  23 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que representa el estado de las interfaces seriales y Ethernet
* del ruteador Cisco 2501.
*/

import java.util.HashMap;
import java.util.Iterator;
import java.util.StringTokenizer;

import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.framework.avh.Resource;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.ciscorouter.agent.SNMPConstants;
import com.networkagent.ciscorouter.mib.MIBConstants;
import com.networkagent.ciscorouter.avh.TelnetManager;
import com.networkagent.ciscorouter.avh.AVHConstants;

public class RouterInterface extends Resource
{
   private String   telnetCommand;
   private HashMap  varBindMap;

   public RouterInterface(String id, int index, String description, String hardType, String
telnetCommand) {
       super(id);
       VariablesBind varBind;

       this.varBindMap = new HashMap();

       varBind = new VariablesBind();
       varBind.setDataLength(0x04);
       varBind.setDataType(SNMPConstants.SNMP_INTEGER);
       varBind.setInstance(MIBConstants.ifIndex);
       varBind.setValue(new Integer(index));
       this.varBindMap.put(varBind.getInstance(), varBind);

       varBind = new VariablesBind();
       varBind.setDataLength(description.length());
       varBind.setDataType(SNMPConstants.SNMP_STRING);
       varBind.setInstance(MIBConstants.locIfDescr);
       varBind.setValue(description);
       this.varBindMap.put(varBind.getInstance(), varBind);

       varBind = new VariablesBind();
       varBind.setDataLength(0x04);
       varBind.setDataType(SNMPConstants.SNMP_INTEGER);
       varBind.setInstance(MIBConstants.locIfCollisions);
       varBind.setValue(new Integer(0));
       this.varBindMap.put(varBind.getInstance(), varBind);

       varBind = new VariablesBind();
       varBind.setDataLength(0x04);
       varBind.setDataType(SNMPConstants.SNMP_INTEGER);
       varBind.setInstance(MIBConstants.locIfKeep);
       varBind.setValue(new Integer(0));
```

```
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(hardType.length());
varBind.setDataType(SNMPConstants.SNMP_STRING);
varBind.setInstance(MIBConstants.locIfHardType);
varBind.setValue(hardType);
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInAbort);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfLineProt);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfResets);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInRunts);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInGiants);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInCRC);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInFrame);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInOverrun);
varBind.setValue(new Integer(0));
this.varBindMap.put(varBind.getInstance(), varBind);

varBind = new VariablesBind();
varBind.setDataLength(0x04);
varBind.setDataType(SNMPConstants.SNMP_INTEGER);
varBind.setInstance(MIBConstants.locIfInIgnored);
varBind.setValue(new Integer(0));
```

```
        this.varBindMap.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        varBind.setInstance(MIBConstants.locIfipInPkts);
        varBind.setValue(new Integer(0));
        this.varBindMap.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        varBind.setInstance(MIBConstants.locIfipOutPkts);
        varBind.setValue(new Integer(0));
        this.varBindMap.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        varBind.setInstance(MIBConstants.locIfipInOctects);
        varBind.setValue(new Integer(0));
        this.varBindMap.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        varBind.setInstance(MIBConstants.locIfipOutOctects);
        varBind.setValue(new Integer(0));
        this.varBindMap.put(varBind.getInstance(), varBind);

        varBind = new VariablesBind();
        varBind.setDataLength(0x04);
        varBind.setDataType(SNMPConstants.SNMP_INTEGER);
        varBind.setInstance(MIBConstants.locIfLineInt);
        varBind.setValue(new Integer(0));
        this.varBindMap.put(varBind.getInstance(), varBind);

        this.telnetCommand = telnetCommand;
    }

    public boolean getState() {
        TelnetManager    telnetManager;
        AgentResources   agentResources;
        VariablesBind varBind;
        String       interfaceState;
        boolean          stateChanged = false;

        agentResources    = AgentResources.getInstance();
        telnetManager     = (TelnetManager)
agentResources.getResource(AVHConstants.TELNET_MANAGER);
        interfaceState    = telnetManager.sendCommand(this.telnetCommand);

        if (interfaceState == null) {
            Iterator iterator;

            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfCollisions);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInAbort);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfKeep);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfLineProt);
            varBind.setValue(new Integer(0));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfResets);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInRunts);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInGiants);
            varBind.setValue(new Integer(-1));
            varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInCRC);
```

```
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInFrame);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInOverrun);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfInIgnored);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfipInPkts);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfipOutPkts);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfipInOctects);
    varBind.setValue(new Integer(-1));
    varBind   = (VariablesBind) this.varBindMap.get(MIBConstants.locIfipOutOctects);
    varBind.setValue(new Integer(-1));

    telnetManager.connect("200.34.38.1", "Cisco2501", "router", "router");
    stateChanged = true;
} else {
    StringTokenizer lineTokenizer;
    StringTokenizer spaceTokenizer;
    String       line;
    String       data;
    String       format;

    lineTokenizer  = new StringTokenizer(interfaceState,"\n");
    if (lineTokenizer.hasMoreTokens()) {
       line         = lineTokenizer.nextToken();
       spaceTokenizer   = new StringTokenizer(line, " ");

       // Tipo y Descripción de la Interfaz.
       data   = spaceTokenizer.nextToken();
       format = data.substring(0,data.length()-1);
       varBind   = (VariablesBind) varBindMap.get(MIBConstants.locIfHardType);
       varBind.setValue(format);
       varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfDescr);
       varBind.setValue(data);

       varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfLineInt);
       while(!(data.equals("up") || data.equals("down"))) {
          data = spaceTokenizer.nextToken();
       }
       if (data.equals("up")) {
          if ( ((Integer)varBind.getValue()).intValue() != 1 ) {
             stateChanged = true;
          }
          varBind.setValue(new Integer(1));
       } else {
          if ( ((Integer)varBind.getValue()).intValue() != 0 ) {
             stateChanged = true;
          }
          varBind.setValue(new Integer(0));
       }

       varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfLineProt);
       while(!(data.equals("up") || data.equals("down"))) {
          data = spaceTokenizer.nextToken();
       }
       if (data.equals("up")) {
          if ( ((Integer)varBind.getValue()).intValue() != 1 ) {
             stateChanged = true;
          }
          varBind.setValue(new Integer(1));
       } else {
          if ( ((Integer)varBind.getValue()).intValue() != 0 ) {
             stateChanged = true;
          }
          varBind.setValue(new Integer(0));
       }
```

```
// Keepalive
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfKeep);
while(! line.trim().startsWith("Keepalive")) {
    line = lineTokenizer.nextToken();
}
spaceTokenizer = new StringTokenizer(line.trim(), " ");
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if (data.equals("set")) {
    if ( ((Integer)varBind.getValue()).intValue() != 1 ) {
        stateChanged = true;
    }
    varBind.setValue(new Integer(1));
} else {
    if ( ((Integer)varBind.getValue()).intValue() != 0 ) {
        stateChanged = true;
    }
    varBind.setValue(new Integer(0));
}

while(line.trim().indexOf("output rate") == -1) {
    line = lineTokenizer.nextToken();
}
line = lineTokenizer.nextToken();
spaceTokenizer = new StringTokenizer(line.trim(), " ");

// IP Input Packets
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfipInPkts);
data   = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
       stateChanged = true;
}
varBind.setValue(new Integer(data));

// IP Input Octects
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfipInOctects);
spaceTokenizer.nextToken();
spaceTokenizer.nextToken();
data    = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
       stateChanged = true;
}
varBind.setValue(new Integer(data));

line = lineTokenizer.nextToken();
spaceTokenizer = new StringTokenizer(line.trim(), " ");

// In Runts
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInRunts);
spaceTokenizer.nextToken();
spaceTokenizer.nextToken();
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
       stateChanged = true;
}
varBind.setValue(new Integer(data));

// In Giants
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInGiants);
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
       stateChanged = true;
}
varBind.setValue(new Integer(data));

// In Giants
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInGiants);
spaceTokenizer.nextToken();
```

```
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

// Aborts
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInAbort);
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

line = lineTokenizer.nextToken();
spaceTokenizer = new StringTokenizer(line.trim(), " ");

// CRC
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInCRC);
spaceTokenizer.nextToken();
spaceTokenizer.nextToken();
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

// Frame
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInFrame);
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

// Overrun
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInOverrun);
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

// Ignored
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfInIgnored);
spaceTokenizer.nextToken();
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

while (line.indexOf("packets output") == -1) {
   line = lineTokenizer.nextToken();
}
spaceTokenizer = new StringTokenizer(line.trim(), " ");

// IP Output Packets
varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfipOutPkts);
data = spaceTokenizer.nextToken();
if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
      stateChanged = true;
}
varBind.setValue(new Integer(data));

// IP Output Octects
```

```
                varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfipOutOctects);
                spaceTokenizer.nextToken();
                spaceTokenizer.nextToken();
                data = spaceTokenizer.nextToken();
                if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
                      stateChanged = true;
                }
                varBind.setValue(new Integer(data));

                line = lineTokenizer.nextToken();
                spaceTokenizer = new StringTokenizer(line.trim(), " ");

                // Collisions
                varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfCollisions);
                spaceTokenizer.nextToken();
                spaceTokenizer.nextToken();
                spaceTokenizer.nextToken();
                data = spaceTokenizer.nextToken();
                if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
                      stateChanged = true;
                }
                varBind.setValue(new Integer(data));

                // Resets
                varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfResets);
                spaceTokenizer.nextToken();
                data = spaceTokenizer.nextToken();
                if ( ((Integer)varBind.getValue()).intValue() != Integer.parseInt(data) ) {
                      stateChanged = true;
                }
                varBind.setValue(new Integer(data));
             }
          }

          return stateChanged;
       }

       public Object getValue() {
          return this.varBindMap;
       }

       public void setValue(Object value) {
          this.varBindMap = (HashMap) value;
       }
    }
```

D.4.4 RouterResourcesContainer

```
/**
 * @(#)com.networkagent.ciscorouter.avh.RouterResourcesContainer.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  26 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que implementa el contenedor de recursos de la AVH. En este caso
* se encarga de crear los tres tipos de interfaces del ruteador, así como
* de enviar el cambio de estado en el cual se indica que el Agente
* se ha inicializado.
*/

import java.util.HashMap;
import java.util.Vector;
import java.util.Iterator;

import com.networkagent.framework.avh.ResourcesContainer;
import com.networkagent.framework.mib.MIBManager;
import com.networkagent.framework.util.AgentResources;
import com.networkagent.framework.util.AgentResourcesKeys;
import com.networkagent.framework.agent.MessageQueue;
import com.networkagent.framework.avh.ReportCreator;
import com.networkagent.ciscorouter.avh.TelnetManager;
import com.networkagent.ciscorouter.avh.RouterInterface;
import com.networkagent.ciscorouter.mib.MIBConstants;
import com.networkagent.ciscorouter.agent.SNMPTrapMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;

public class RouterResourcesContainer extends ResourcesContainer
{
   public RouterResourcesContainer() {
      super();
   }

   public void initialize() {
      TelnetManager    telnetManager;
      AgentResources   agentResources;
      RouterInterface  routerInterface;
      MIBManager       mibManager;
      ReportCreator reportCreator;
      Vector        vector;
      Iterator      iterator;
      MessageQueue mibQueue;
      SNMPTrapMessage trapMessage;

      agentResources   = AgentResources.getInstance();
      telnetManager    = (TelnetManager)
agentResources.getResource(AVHConstants.TELNET_MANAGER);
      mibManager       = (MIBManager)
agentResources.getResource(AgentResourcesKeys.MIB_MANAGER);
      reportCreator= (ReportCreator)
agentResources.getResource(AgentResourcesKeys.REPORT_CREATOR);
      mibQueue     = (MessageQueue)
agentResources.getResource(AgentResourcesKeys.MIB_QUEUE);

      telnetManager.connect("200.34.38.1", "Cisco2501", "router", "router");

      routerInterface = new RouterInterface(MIBConstants.ifIndex + ".1", 1, "Ethernet0",
"Ethernet", "show int e0");
      vector       = new Vector();
```

```
       iterator     = ((HashMap) routerInterface.getValue()).values().iterator();
       while(iterator.hasNext()) {
          vector.add(iterator.next());
       }
       mibManager.insertManagedObject(vector);
       super.insertResource(routerInterface);

       routerInterface = new RouterInterface(MIBConstants.ifIndex + ".2", 2, "Serial0",
"Serial",   "show int s0");
       vector       = new Vector();
       iterator     = ((HashMap) routerInterface.getValue()).values().iterator();
       while(iterator.hasNext()) {
          vector.add(iterator.next());
       }
       mibManager.insertManagedObject(vector);
       super.insertResource(routerInterface);

       routerInterface = new RouterInterface(MIBConstants.ifIndex + ".3", 3, "Serial1",
"Serial",   "show int s1");
       vector       = new Vector();
       iterator     = ((HashMap) routerInterface.getValue()).values().iterator();
       while(iterator.hasNext()) {
          vector.add(iterator.next());
       }
       mibManager.insertManagedObject(vector);
       super.insertResource(routerInterface);

       trapMessage = new SNMPTrapMessage();
       trapMessage.setEnterprise(MIBConstants.lifEntry);
       trapMessage.setGenericTrap(SNMPConstants.SNMP_WARMSTART);
       trapMessage.setAgentAddress((String)
agentResources.getResource(AgentResourcesKeys.AGENT_IP));
       trapMessage.setEnterprise(MIBConstants.lifTable);
       trapMessage.setTimeStamp(0);
       trapMessage.setVarBindList(null);

       mibQueue.insertMessage(trapMessage);


   }
}
```

D.4.5 StatusPeer

```
/**
 * @(#)com.networkagent.ciscorouter.avh.StatusPeer.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  27 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Interfaz utilizada para conocer el estado de una conexión Telnet.
*/

import java.util.Vector;

public interface StatusPeer {
   public Object notifyStatus(Vector status);
}
```

D.4.6 TelnetIO

```
/**
 * @(#)com.networkagent.ciscorouter.avh.TelnetIO.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  27 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que implementa el mecanismo entrada/salida de información a
* través de Telnet.
*/


import java.net.Socket;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.awt.Dimension;
import java.util.Vector;

public class TelnetIO implements StatusPeer
{
   private static int debug = 0;
   private byte neg_state  = 0;
   private final static byte STATE_DATA       = 0;
   private final static byte STATE_IAC           = 1;
   private final static byte STATE_IACSB         = 2;
   private final static byte STATE_IACWILL       = 3;
   private final static byte STATE_IACDO         = 4;
   private final static byte STATE_IACWONT       = 5;
   private final static byte STATE_IACDONT       = 6;
   private final static byte STATE_IACSBIAC      = 7;
   private final static byte STATE_IACSBDATA     = 8;
   private final static byte STATE_IACSBDATAIAC  = 9;
   private byte current_sb;
   private final static byte IAC           = (byte)255;
   private final static byte EOR           = (byte)239;
   private final static byte WILL      = (byte)251;
   private final static byte WONT      = (byte)252;
   private final static byte DO        = (byte)253;
   private final static byte DONT      = (byte)254;
   private final static byte SB            = (byte)250;
   private final static byte SE            = (byte)240;
   private final static byte TELOPT_ECHO   = (byte)1;
   private final static byte TELOPT_EOR    = (byte)25;
   private final static byte TELOPT_NAWS   = (byte)31;
   private final static byte TELOPT_TTYPE  = (byte)24;
   private final static byte[] IACWILL     = { IAC, WILL };
   private final static byte[] IACWONT     = { IAC, WONT };
   private final static byte[] IACDO       = { IAC, DO  };
   private final static byte[] IACDONT     = { IAC, DONT };
   private final static byte[] IACSB       = { IAC, SB };
   private final static byte[] IACSE       = { IAC, SE };
   private final static byte TELQUAL_IS = (byte)0;
   private final static byte TELQUAL_SEND = (byte)1;
    private byte[] receivedDX;
   private byte[] receivedWX;
   private byte[] sentDX;
   private byte[] sentWX;
   private Socket socket;
   private BufferedInputStream is;
   private BufferedOutputStream os;
```

```
private StatusPeer peer = this;

public void connect(String address, int port) throws IOException {
   if(debug > 0) {
      System.out.println("Telnet.connect("+address+","+port+")");
   }

   socket       = new Socket(address, port);
   is        = new BufferedInputStream(socket.getInputStream());
   os        = new BufferedOutputStream(socket.getOutputStream());
   neg_state    = 0;
   receivedDX   = new byte[256];
   sentDX       = new byte[256];
   receivedWX   = new byte[256];
   sentWX       = new byte[256];
}

public void disconnect() throws IOException {
   if(debug > 0) {
      System.out.println("TelnetIO.disconnect()");
   }

   if(socket !=null) {
      socket.close();
   }
}

public void connect(String address) throws IOException {
   connect(address, 23);
}

public void setPeer(StatusPeer obj) {
   peer = obj;
}

public int available() throws IOException {
   return is.available();
}

public byte[] receive() throws IOException {
   int count    = is.available();
   byte buf[]   = new byte[count];

   count     = is.read(buf);
   if(count < 0) {
      throw new IOException("Connection closed.");
   }

   if(debug > 1) {
      System.out.println("TelnetIO.receive(): read bytes: "+count);
   }

   buf = negotiate(buf, count);
   return buf;
}

public void send(byte[] buf) throws IOException {
   if(debug > 1) {
      System.out.println("TelnetIO.send("+buf+")");
   }

   os.write(buf);
   os.flush();
}

public void send(byte b) throws IOException {
   if(debug > 1) {
      System.out.println("TelnetIO.send("+b+")");
   }
```

```
   os.write(b);
   os.flush();
}

private void handle_sb(byte type, byte[] sbdata, int sbcount) throws IOException {
   if(debug > 1) {
      System.out.println("TelnetIO.handle_sb("+type+")");
   }

   switch (type) {
      case TELOPT_TTYPE:
         if (sbcount>0 && sbdata[0]==TELQUAL_SEND) {
            String ttype;

            send(IACSB);send(TELOPT_TTYPE);send(TELQUAL_IS);
            Vector vec = new Vector(2);
            vec.addElement("TTYPE");
            ttype = (String)peer.notifyStatus(vec);

         if(ttype == null) {
            ttype = "dumb";
         }

         byte[] bttype = new byte[ttype.length()];

         ttype.getBytes(0,ttype.length(), bttype, 0);
         send(bttype);
         send(IACSE);
      }
   }
}

public Object notifyStatus(Vector status) {
   if(debug > 0) {
      System.out.println("TelnetIO.notifyStatus("+status+")");
   }

   return null;
}

private byte[] negotiate(byte buf[], int count) throws IOException {
   byte   nbuf[]       = new byte[count];
   byte   sbbuf[]   = new byte[count];
   byte   sendbuf[]    = new byte[3];
   byte   b;
   byte   reply;
   int    sbcount   = 0;
   int    boffset   = 0;
   int    noffset   = 0;
   Vector vec       = new Vector(2);


   if(debug > 1) {
      System.out.println("TelnetIO.negotiate("+buf+","+count+")");
   }


   while(boffset < count) {
      b=buf[boffset++];

      if (b>=128) {
         b=(byte)((int)b-256);
      }

      switch (neg_state) {
         case STATE_DATA:
            if (b==IAC) {
               neg_state = STATE_IAC;
            } else {
               nbuf[noffset++]=b;
```

```
    }
    break;

case STATE_IAC:
    switch (b) {
        case IAC:
            if(debug > 2) {
                System.out.print("IAC ");
            }

            neg_state = STATE_DATA;
            nbuf[noffset++]=IAC;
        break;

        case WILL:
            if(debug > 2) {
                System.out.print("WILL ");
            }

            neg_state = STATE_IACWILL;
            break;

        case WONT:
            if(debug > 2) {
                System.out.print("WONT ");
            }

            neg_state = STATE_IACWONT;
            break;

        case DONT:
            if(debug > 2) {
                System.out.print("DONT ");
            }

            neg_state = STATE_IACDONT;
            break;

        case DO:
            if(debug > 2) {
                System.out.print("DO ");
            }

            neg_state = STATE_IACDO;
            break;

        case EOR:
            if(debug > 2) {
                System.out.print("EOR ");
            }

            neg_state = STATE_DATA;
            break;

        case SB:
            if(debug > 2) {
                System.out.print("SB ");
            }

            neg_state = STATE_IACSB;
            sbcount = 0;
            break;

        default:
            if(debug > 2) {
                System.out.print("<UNKNOWN "+b+" > ");
            }

            neg_state = STATE_DATA;
            break;
```

```
            }
        break;

    case STATE_IACWILL:
        switch(b) {
            case TELOPT_ECHO:
                if(debug > 2) {
                    System.out.println("ECHO");
                }

                reply = DO;
                vec = new Vector(2);
                vec.addElement("NOLOCALECHO");
                peer.notifyStatus(vec);
                break;

            case TELOPT_EOR:
                if(debug > 2)
                    System.out.println("EOR");
                reply = DO;
                break;

            default:
                if(debug > 2) {
                    System.out.println("<UNKNOWN,"+b+">");
                }

                reply = DONT;
                break;
        }

        if(debug > 1) {
            System.out.println("<"+b+", WILL ="+WILL+">");
        }

        if (reply != sentDX[b+128] || WILL != receivedWX[b+128]) {
            sendbuf[0] = IAC;
            sendbuf[1] = reply;
            sendbuf[2] = b;

            send(sendbuf);
            sentDX[b+128]     = reply;
            receivedWX[b+128] = WILL;
        }

        neg_state = STATE_DATA;
        break;

    case STATE_IACWONT:
        switch(b) {
            case TELOPT_ECHO:
                if(debug > 2) {
                    System.out.println("ECHO");
                }

                vec = new Vector(2);
                vec.addElement("LOCALECHO");
                peer.notifyStatus(vec);
                reply = DONT;
                break;

            case TELOPT_EOR:
                if(debug > 2) {
                    System.out.println("EOR");
                }

                reply = DONT;
                break;

            default:
```

```
            if(debug > 2) {
               System.out.println("<UNKNOWN,"+b+">");
            }

            reply = DONT;
            break;
      }

      if (reply != sentDX[b+128] || WONT != receivedWX[b+128]) {
         sendbuf[0] = IAC;
         sendbuf[1] = reply;
         sendbuf[2] = b;
         send(sendbuf);
         sentDX[b+128]      = reply;
         receivedWX[b+128] = WILL;
      }

      neg_state = STATE_DATA;
      break;

   case STATE_IACDO:
      switch (b) {
         case TELOPT_ECHO:
            if(debug > 2) {
               System.out.println("ECHO");
            }

            reply = WILL;
            vec = new Vector(2);
            vec.addElement("LOCALECHO");
            peer.notifyStatus(vec);
            break;

         case TELOPT_TTYPE:
            if(debug > 2)
               System.out.println("TTYPE");
            reply = WILL;
            break;

         case TELOPT_NAWS:
            if(debug > 2) {
               System.out.println("NAWS");
            }

            vec = new Vector(2);
            vec.addElement("NAWS");
            Dimension size = (Dimension)
            peer.notifyStatus(vec);
            receivedDX[b] = DO;

         if(size == null) {
            send(IAC);
            send(WONT);
            send(TELOPT_NAWS);
            reply = WONT;
            sentWX[b] = WONT;
            break;
         }

         reply     = WILL;
         sentWX[b]  = WILL;
         sendbuf[0] = IAC;
         sendbuf[1] = WILL;
         sendbuf[2] = TELOPT_NAWS;
         send(sendbuf);
         send(IAC);send(SB);send(TELOPT_NAWS);
         send((byte) (size.width >> 8));
         send((byte) (size.width & 0xff));
         send((byte) (size.height >> 8));
         send((byte) (size.height & 0xff));
```

```
            send(IAC);send(SE);
            break;

        default:
            if(debug > 2) {
                System.out.println("<UNKNOWN,"+b+">");
            }

            reply = WONT;
            break;
        }

        if (reply != sentWX[128+b] || DO != receivedDX[128+b]) {
            sendbuf[0] = IAC;
            sendbuf[1] = reply;
            sendbuf[2] = b;
            send(sendbuf);
            sentWX[b+128]     = reply;
            receivedDX[b+128] = DO;
        }

        neg_state = STATE_DATA;
        break;

    case STATE_IACDONT:
        switch (b) {
            case TELOPT_ECHO:
                if(debug > 2) {
                    System.out.println("ECHO");
                }

                reply  = WONT;
                vec    = new Vector(2);
                vec.addElement("NOLOCALECHO");
                peer.notifyStatus(vec);
                break;

            case TELOPT_NAWS:
                if(debug > 2) {
                    System.out.println("NAWS");
                }

                reply  = WONT;
                break;

            default:
                if(debug > 2) {
                    System.out.println("<UNKNOWN,"+b+">");
                }

                reply  = WONT;
                break;
        }

        if (reply != sentWX[b+128] || DONT != receivedDX[b+128]) {
            send(IAC);
            send(reply);
            send(b);
            sentWX[b+128]    = reply;
            receivedDX[b+128]   = DONT;
        }

        neg_state = STATE_DATA;
        break;

    case STATE_IACSBIAC:
        if(debug > 2) {
            System.out.println(""+b+" ");
        }
```

```
    if (b == IAC) {
       sbcount      = 0;
       current_sb   = b;
       neg_state = STATE_IACSBDATA;
    } else {
       System.out.println("(bad) "+b+" ");
       neg_state = STATE_DATA;
    }
    break;

case STATE_IACSB:
    if(debug > 2) {
       System.out.println(""+b+" ");
    }

    switch (b) {
       case IAC:
          neg_state = STATE_IACSBIAC;
          break;
       default:
          current_sb   = b;
          sbcount      = 0;
          neg_state = STATE_IACSBDATA;
          break;
    }
    break;

case STATE_IACSBDATA:
    if (debug > 2) {
       System.out.println(""+b+" ");
    }

    switch (b) {
       case IAC:
          neg_state = STATE_IACSBDATAIAC;
          break;

       default:
          sbbuf[sbcount++] = b;
          break;
    }

    break;

case STATE_IACSBDATAIAC:
    if (debug > 2) {
       System.out.println(""+b+" ");
    }

    switch (b) {
       case IAC:
          neg_state = STATE_IACSBDATA;
          sbbuf[sbcount++] = IAC;
          break;

       case SE:
          handle_sb(current_sb,sbbuf,sbcount);
          current_sb   = 0;
          neg_state = STATE_DATA;
          break;

       case SB:
          handle_sb(current_sb,sbbuf,sbcount);
          neg_state = STATE_IACSB;
          break;

       default:
          neg_state = STATE_DATA;
          break;
    }
```

```
                break;

            default:
                if (debug > 2) {
                    System.out.println("Esto no debería ocurrir: " + neg_state + " ");
                }

                neg_state = STATE_DATA;
                break;
            }
        }

        buf = new byte[noffset];

        System.arraycopy(nbuf, 0, buf, 0, noffset);
        return buf;
    }
}
```

D.4.7 TelnetManager

```
/**
 * @(#)com.networkagent.ciscorouter.avh.TelnetManager.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  28 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que permite conectarse al ruteador Cisco 2501 a través de Telnet.
*/

import java.io.IOException;

import com.networkagent.ciscorouter.avh.TelnetWrapper;

public class TelnetManager {
   private TelnetWrapper telnet;

   public void connect(String ip, String netElementName, String defPwd, String privPwd) {
      try {
         this.telnet = new TelnetWrapper(ip);

         this.telnet.setPrompt(netElementName + ">");
         this.telnet.sendLine(defPwd);
         this.telnet.setPrompt(netElementName + "#");
         this.telnet.sendLine("enable\n" + privPwd);
      } catch (IOException ioexc)   {
         System.out.println ("Error de IO. Excepción: " + ioexc.toString());
      }
   }

   public String sendCommand(String command) {
      String result = null;

      try {
         if (this.telnet != null) {
            result = this.telnet.sendLine(command);
         }
      } catch (IOException ioexc)   {
         System.out.println ("Error de IO. Excepción: " + ioexc.toString());
      }

      return result;
   }

   public void disconnect() {
      try {
         this.telnet.setPrompt("");
         this.telnet.send("exit");
         this.telnet.disconnect();
      } catch (IOException ioexc)   {
         System.out.println ("Error de IO. Excepción: " + ioexc.toString());
      }
   }
}
```

D.4.8 TelnetWrapper

```
/**
 * @(#)com.networkagent.ciscorouter.avh.TelnetWrapper.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  28 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que encapsula los mecanismo de envío y recepción de datos
* a través de Telnet.
*/

import java.io.IOException;
import java.util.Date;

public class TelnetWrapper {
   private TelnetIO       tio;
   public  boolean     debug = false;
   private String         prompt;
   private static String  defaultPrompt = "$ ";
   private static String  defaultLogin = null;
   private static String  defaultPassword = null;

   public void wait(String token) throws IOException {
      wait(token, -1);
   }

   public void wait(String token, long timeout) throws IOException, TimedOutException {
      String tmp       = "";
      long   deadline = 0;

      if(debug) {
         System.out.println("wait(" + token + ", " + timeout + ")...");
      }
       if(timeout >= 0) {
         deadline = new Date().getTime() + timeout;
      }
   do {
          if(timeout >= 0) {
             while(available() <= 0) {
                if(new Date().getTime() > deadline) {
                   throw new TimedOutException();
                }
                try {
                   Thread.currentThread().sleep(100);
                } catch(InterruptedException intexc) {
                   System.out.println ("Error de interrupción. Excepción: " +
intexc.toString());
                }
             }
          }
      tmp = receive();
      } while(tmp.indexOf(token) == -1);

      if(debug) {
         System.out.println("wait(" + token  + ", " + timeout + ") successful.");
      }
   }

   public int available() throws IOException {
      return tio.available();
   }
```

```
   public String receive() throws IOException {
      String s = new String(receiveBytes(), 0);

      if(debug) {
         System.out.println(s);
      }

      return s;
   }

   public byte[] receiveBytes() throws IOException {
      return tio.receive();
   }

   public String receiveUntil(String token) throws IOException {
      return receiveUntil(token, -1);
   }

   public String receiveUntil(String token, long timeout) throws IOException,
TimedOutException {
      StringBuffer buf = new StringBuffer();
      long deadline    = 0;

      if(timeout >= 0) {
         deadline = new Date().getTime() + timeout;
      }
      do {
         if(timeout >= 0) {
            while(available() <= 0){
               if(new Date().getTime() > deadline) {
                  throw new TimedOutException();
               }
               try {
                  Thread.currentThread().sleep(100);
               } catch(InterruptedException intexc) {
                  System.out.println ("Error de Interrupción. Excepción: " +
intexc.toString());
               }
            }
         }

         buf.append(receive());
      } while(buf.toString().indexOf(token) == -1);

      return buf.toString();
   }

   public void send(String s) throws IOException {
      byte[] buf = new byte[s.length()];

      if(debug) {
         System.out.println(s);
      }

      s.getBytes(0, buf.length, buf, 0);
      tio.send(buf);
   }

   public String sendLine(String command) throws IOException {
      if(command.charAt(command.length() -1) != '\r') {
         command += "\r";
      }
      send(command);
      String s = receiveUntil(prompt);
      return s.substring(command.length() + 1, s.indexOf(prompt));
   }

   public void send(byte[] buf) throws IOException {
      tio.send(buf);
   }
```

```
    public void login(String loginName, String password) throws IOException {
       wait("login:");
       send(loginName + "\r");
       wait("Password:");
       sendLine(password + "\r");
    }

    public TelnetWrapper(String host) throws IOException {
       tio = new TelnetIO();
       setPrompt(defaultPrompt);
       tio.connect(host);
       if(defaultLogin != null && defaultPassword != null) {
          login(defaultLogin, defaultPassword);
       }
    }

    public TelnetWrapper(String host, int port) throws IOException {
        tio = new TelnetIO();
        setPrompt(defaultPrompt);
       tio.connect(host, port);

       if(defaultLogin != null && defaultPassword != null) {
          login(defaultLogin, defaultPassword);
       }
    }

    public void setPrompt(String prompt) {
       if(prompt == null) {
          throw new IllegalArgumentException("null prompt.");
       }

       this.prompt = prompt;
    }

    public static void setDefaultPrompt(String prompt) {
       if(prompt == null) {
          throw new IllegalArgumentException("null prompt.");
       }

       defaultPrompt = prompt;
    }

    public static void setLogin(String login, String password) {
       if(login == null || password == null) {
          throw new IllegalArgumentException("null login or password.");
       }

       defaultLogin = login;
       defaultPassword = password;
    }

    public static void unsetLogin() {
       defaultLogin = defaultPassword = null;
    }

    public void disconnect() throws IOException {
       if(tio != null) {
          tio.disconnect();
       }
       tio = null;
    }

    public void finalize() {
       try {
          disconnect();
       } catch(IOException ioexc) {
          System.out.println ("Error de IO. Excepción: " + ioexc.toString());
       }
    }
}
```

D.4.9 TimeOutException

```
/**
 * @(#)com.networkagent.ciscorouter.avh.TelnetManager.java
 *
 * @author:      Carlos Omar Torres González
 * @Creation Date:  26 de Diciembre de 2002
 * @Version:     1.0
 * @Comment:     PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Excepción de time-out en una conexión de Telnet.
*/

import java.io.IOException;

public class TimedOutException extends IOException
{
   public TimedOutException() {
   }

   public TimedOutException(String message) {
      super(message);
   }
}
```

D.4.10 RandomResourcesIterator

```
/**
 * @(#)com.networkagent.ciscorouter.avh.RandomResourcesIterator.java
 *
 * @author:     Carlos Omar Torres González
 * @Creation Date:  28 de Diciembre de 2002
 * @Version:    1.0
 * @Comment:    PROPIEDAD INTELECTUAL.
 */

package com.networkagent.ciscorouter.avh;

/**
* Clase que permite crear reportes de cambio de estado en los
* Recursos del elemento de red.
*/

import java.util.HashMap;
import java.util.Vector;
import java.util.Iterator;
import java.util.Date;

import com.networkagent.framework.avh.ReportCreator;
import com.networkagent.framework.avh.Resource;
import com.networkagent.framework.agent.ReportMessage;
import com.networkagent.framework.agent.VariablesBind;
import com.networkagent.ciscorouter.agent.SNMPTrapMessage;
import com.networkagent.ciscorouter.agent.SNMPConstants;
import com.networkagent.ciscorouter.mib.MIBConstants;
import com.networkagent.ciscorouter.avh.AVHConstants;

public class TrapReportCreator implements ReportCreator
{
   int    reportCounter;
   long   timeCreation;

   public TrapReportCreator() {
      super();
      this.reportCounter = 0;
      this.timeCreation  = new Date().getTime();
   }

   public ReportMessage createReport(Resource resource) {
      SNMPTrapMessage  reportMessage;
      HashMap      varBindMap;
      Vector       varBindList;
      Iterator     iterator;
      VariablesBind varBind;
      String       instance = "";

      reportMessage    = new SNMPTrapMessage();
      varBindList      = new Vector();
      varBindMap       = (HashMap) resource.getValue();
      iterator     = varBindMap.values().iterator();

      while(iterator.hasNext()) {
         varBind      = (VariablesBind) iterator.next();
         instance  = varBind.getInstance();
         varBindList.add(varBind);
      }

      reportMessage.setVarBindList(varBindList);
      reportMessage.setMessageId(String.valueOf(++this.reportCounter));
      reportMessage.setTimeStamp(new Date().getTime() - this.timeCreation);

      if (instance.startsWith(MIBConstants.lifEntry)) {
         reportMessage.setEnterprise(MIBConstants.lifEntry);
```

```
            varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfCollisions);
            if ( ((Integer)varBind.getValue()).intValue() == -1 ) {
               reportMessage.setSpecificTrap(AVHConstants.FATAL_LINE_ERROR);
               reportMessage.setGenericTrap(SNMPConstants.SNMP_LINKDOWN);
            } else {
               varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfLineInt);
               if ( ((Integer)varBind.getValue()).intValue() == 0 ) {
                  reportMessage.setSpecificTrap(AVHConstants.LINE_INT_DOWN);
                  reportMessage.setGenericTrap(SNMPConstants.SNMP_LINKDOWN);
               } else {
                  varBind = (VariablesBind) varBindMap.get(MIBConstants.locIfLineProt);
                  if ( ((Integer)varBind.getValue()).intValue() == 0 ) {
                     reportMessage.setSpecificTrap(AVHConstants.LINE_PROT_DOWN);
                     reportMessage.setGenericTrap(SNMPConstants.SNMP_LINKDOWN);
                  } else {
                     reportMessage.setSpecificTrap(AVHConstants.CHANGE_VALUES);
                     reportMessage.setGenericTrap(SNMPConstants.SNMP_ENTERPRISESPECIFIC);
                  }
               }
            }
         }

         return reportMessage;
      }
   }
```

Bibliografía

[Abbot, 1983] Abbott, R.; *Program Design by Informal English Descriptions*; Communications of the ACM vol.26(11); U.S.A.; 1983.

[Aidarous, 1998] Aidarous, S., Plevyak, T.; *Telecommunications Network Management. Technologies and Implementations*; Wiley-IEEE Press; U.S.A.; 1998.

[Alur, 2001] Alur, D., Crupi, J., Malks, D.; *Core J2ee Patterns. Best Practices and Design Strategies*; Prentice Hall; U.S.A.; 2001

[Black, 1992] Black, U.; *Network Management Standards*; McGraw Hill; U.S.A.; 1995.

[CCITT X.710, 1992] International Telegraph and Telephone Consultative Committee (CCITT); *Recommendation X.710 Common Management Information Service Element*; CCITT; Switzerland; 1992.

[CCITT X.711, 1992] International Telegraph and Telephone Consultative Committee (CCITT); *Recommendation X.711 Common Management Information Protocol Specification*; CCITT; Switzerland; 1992.

[CCITT X.721, 1992] International Telegraph and Telephone Consultative Committee (CCITT); *Recommendation X.721 Definition of Management Information*; CCITT; Switzerland; 1992.

[Chan, 1998] Chan, S., Lammers, T; Object Oriented Domain Engineering. *5th International Conference on Software Reuse (ICSR5 1998) Tutorial*. IEEE Computer Society Press; U.S.A.; 1998

[CISCO, 1994] Johnson, J.T.; *Cisco Interfaces MIB File*; http://cisco.com ; U.S.A.; 1994.

[Coad, 1995] Coad, P. Yourdon, E.; *Object-Oriented Analysis*.; Yourdon Press; U.S.A.; 1995.

[Ecklund, 1996] E.R. Ecklund, Delcambre, L.M.L., Freiling, M.J.; *Change cases: Use Cases that Identify Future Requirements. Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 342-358; ACM Press; U.S.A.; 1996.

[Fayad, 1999] Fayad, M.E., D. Schmidt, and R. Johnson; *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. Wiley; U.S.A.; 1999.

[Fayad 2, 1999] Fayad, M.E., R. Johnson; *Domain-Specific Application Frameworks: Frameworks Experience by Industry*; Wiley; U.S.A.; 1999.

[Fayad 3, 1999] Fayad, M.E., D. Schmidt, and R. Johnson; *Implementing Application Frameworks: Object-Oriented Framework at Work*. Wiley; U.S.A.; 1999.

[Fayad-Schmidt, 1997] Fayad, M.E., Schmidt, D.; *Object-Oriented Application Frameworks*; Communications of the ACM 40(10); U.S.A.; 1997.

[Gamma, 1995] Gamma, E., Helm, R., Johnson R., Vlissides, J.; *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley Publishing; U.S.A.; 1995.

[Grand, 1998] Grand, M.; *Patterns in Java. A Catalog of Reusable Design Patterns Illustrated with UML*; Wiley; U.S.A.; 1998.

[Hueni, 1995] Hueni, H., Johnson, R., Engel, R.; *A Framework for Network Protocol Software. Proceedings of OOPSLA*; ACM Press; U.S.A.; 1995.

[Johnson-Foote, 1988] Johnson, R.E., Foote, B.; *Designing Reusable Classes*. *Journal of Object Oriented Programming*. ACM Journal; U.S.A.; 1988.

[Kocher, 1999] Kocher, H., Schabernack, J.; A Framework for Network Management Agents. *Domain-Specific Application Frameworks: Frameworks Experience by Industry,* pp 385-396.; Wiley; U.S.A.; 1999.

[Larman, 1998] Larman, C.; *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*; Prentice Hall; U.S.A.; 1998.

[Lewis, 1995] Lewis, T.; *Object-Oriented Application Frameworks*. Greenwich, CT: Manning; U.S.A.; 1995.

[Major, 1997] Major, M.L., McGregor, J.D. *A Qualitative Analysis of Two Requirements Capturing Techniques for Estimating the Size of Object-Oriented Software Projects*. Technical Report; Clemson University; U.S.A.; 1997.

[Miller, 1999] Miller, G.G., McGregor, J.D., Major M.L.; Capturing Framework Requirements. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, pp. 309-321; Wiley; U.S.A.; 1999.

[NMF 037, 1995] Network Management Forum (NMF); *NMF 037 Sub-System Alarm Surveillance Ensemble*; NMF; U.S.A.; 1995.

[Prieto, 1991] Prieto-Díaz, R., Arango, G.; *Domain Analysis and Software Systems Modeling*; IEEE Computer Society Press; U.S.A.; 1991.

[Ranman, 1999] Ranman, L.G.; *Fundamentals of Telecommunications Network Management*; Wiley-IEEE Press; U.S.A.; 1999.

[RFC-1157, 1990] Case, J., Fedor, M., Schoffstall, M., Davin, J.; *A Simple Network Management Protocol (SNMP)*. RFC-1157; U.S.A.; 1992.

[RFC-1213, 1991] McCloghrie, K., Rose, M.T.; *Management Information Base for Network Management of TCP/IP-based internets*; RFC-1213; U.S.A.; 1991.

[Wirfs-Brock, 1990] Wirfs-Brock, R.; Wilkerson, B.; Wiener, L.; *Designing Object-Oriented Software*; Prentice Hall; U.S.A.; 1990.

[Udupa, 1999] Udupa, D.K.; *TMN: Telecommunication Management Networks*; McGraw Hill; U.S.A.; 1999.