GESTIÓN DE REDES DE TELECOMUNICACIONES UTILIZANDO LA PLATAFORMA OSIMIS

Julio César Aristizábal Alzate
Rubén Dario Rojas Pérez

Director
José Luis Arciniegas Herrera

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Conmutación
Popayán
2003

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

CAPITULO I

INTRODUCCIÓN

En la actualidad las redes de telecomunicaciones se han convertido en un pilar fundamental en el desarrollo de la sociedad, y por la misma razón esta exige cada vez mejor desempeño de las redes, por lo que las compañías que brindan los servicios necesitan realizar una gestión que les permita supervisar el comportamiento de la red de una manera eficiente para lograr su mejor desempeño. Por lo general las compañías han ofrecido sus propios sistemas de gestión, pero con la heterogeneidad y globalidad de las redes actuales estas soluciones son incompletas, lo cual conllevo a formación de entidades a nivel internacional que regulen la manera de llevar a cabo la gestión, dando como resultado estándares como el SNMP[1] (Simple Network Management Protocol) y el TMN[2] (Telecommunication Management Network).

Las nuevas tecnologías emergentes como el SDH/SONET[3] (Syncronous Digital Hierarchy/Syncronous Optical Network) en la trasmisión, ATM[4] (Asyncronous Transfer Mode) en conmutación, van a formar la base de la infraestructura de las Telecomunicaciones futuras las cuales poseen unos requerimientos de gestión más complejos. La ITU-T (International Telecommunication Union) en su propósito de estandarización ha elaborado a finales de los años ochenta un conjunto de normas denominadas TMN como una plataforma para la gestión que dé la base conceptual, una arquitectura y unas guías que hicieran posible la construcción de una infraestructura para la gestión de estas tecnologías, mediante la interconexión

de Sistemas de Operación y equipos de Telecomunicación para el intercambio de información de gestión.

Los estándares de TMN abordan tres áreas fundamentales. La primera, una arquitectura con un modelo de niveles: negocio, servicios, red y elementos de red. La segunda, un modelo de información orientado a objetos que especifica nombres comunes y relaciones entre los ítems involucrados en un sistema de gestión. La tercera, un mecanismo de computación distribuida a través del cual se comunican y cooperan los componentes del sistema de gestión. El modelo de niveles TMN y el modelo de información son ampliamente aceptados y utilizados por las empresas operadoras de Telecomunicaciones, no ha sucedido igual con el sistema de computación distribuida propuesto. El desarrollo de éste, al igual que ha sucedido con otros estándares OSI [5] (Open System Interconnection), no ha avanzado mucho desde principios de los 90, mientras que nuevas tecnologías han entrado en escena ofreciendo soluciones efectivas a las necesidades de las aplicaciones distribuidas en entornos heterogéneos, y que cuentan además con un amplio respaldo entre fabricantes y usuarios.

Pero por la misma creciente complejidad y heterogeneidad de las modernas redes de telecomunicaciones se está generando la necesidad de buscar mecanismos simples y uniformes para gestionarlas, y un factor clave en el éxito de esta propuesta consiste en la estrategia de transición entre el estado actual, con una gran profusión de sistemas gestionados por protocolos como SNMP y CMIP/S [6] [7] (Common Management Information Service/Protocol).

La distribución de la monografía se hizo con la intención que los lectores puedan utilizarla para trabajar a la par con la plataforma OSIMIS, por tal razón en este capítulo I se da una ambientación en cuanto a la gestión de redes, para poder entrar en el capítulo II a estudiar la estructura de las plataformas ISODE y OSIMIS, lo cual nos permitirá con éste conocimiento general de las plataformas

concentrarnos en la correcta compilación e instalación de ellas en el capítulo III para luego de tener las plataformas funcionando poder estudiar a fondo todos los servicios y aplicaciones que componen OSIMIS, lo cual presentamos en el capitulo IV, para desembocar así en el mayor aporte que realizamos en el trabajo de grado el cual es la completa documentación de la creación de objetos y agentes gestionados ejemplificándola con un objeto y agente bastante completo que permitirán que futuras implementaciones sean mas sencillas basándose en esta; el capitulo V de esta monografía es entonces el encargado de tratar este tema.

1.1 OBJETIVO

El objetivo de este trabajo de tesis es estudiar la plataforma de gestión OSIMIS [8] (OSI Management Information Services), la cual proporciona los servicios para interconexión entre el mundo OSI y las Aplicaciones de Gestión, brindando la posibilidad de integrarlas con el mundo de la Internet.

OSIMIS es una plataforma de gestión potente y que por tanto amerita un estudio serio de sus capacidades y falencias, por ello inicialmente el trabajo se oriento a su completa especificación y como un segundo paso a cumplir con el objetivo fundamental de incorporar ésta dentro del proyecto GETWeb [9], el cual está en ejecución por el Grupo de Ingeniería Telemática (GIT) de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, y que está concebido a un mayor nivel buscando la construcción de una Infraestructura de Gestión Integrada de Redes y Servicios de Telecomunicaciones soportada en el Web que se apoya en los protocolos de Gestión de Redes (SNMP, CMIP, DMI entre otros) y la potencialidad de las arquitecturas distribuidas donde INTERNET y CORBA [10] van ha ser dos elementos indiscutibles para la integración.

Como el objetivo de GETWeb es crear una plataforma de gestión integrada (ver Figura #1), el papel que cumple OSIMIS dentro de esta es permitir la creación de

objetos gestionados y agentes que se puedan gestionar como dispositivos que cumplan las especificaciones de los protocolos CMIP y SNMP. Además, OSIMIS por medio de la pasarela SNMP/CMIP permite la integración de estos dos dominios de gestión, mientras en el proyecto GETWeb se busca realizar una integración entre la gestión OSI y la infraestructura OMG CORBA.

La plataforma OSIMIS aparece como una solución a muchos de los problemas de gestión actuales en los sistemas de telecomunicaciones y es una plataforma que al estar de acuerdo a las especificaciones del estándar TMN facilita las labores de gestión en todas sus áreas: Fallas, Configuración, Tarificación, Desempeño y Seguridad. Además provee los mecanismos para gestionar dispositivos de redes de telecomunicaciones, sin embargo no es una plataforma completa en el nivel de aplicación, por ello es necesario que se complemente para conseguir un ambiente de gestión integral.

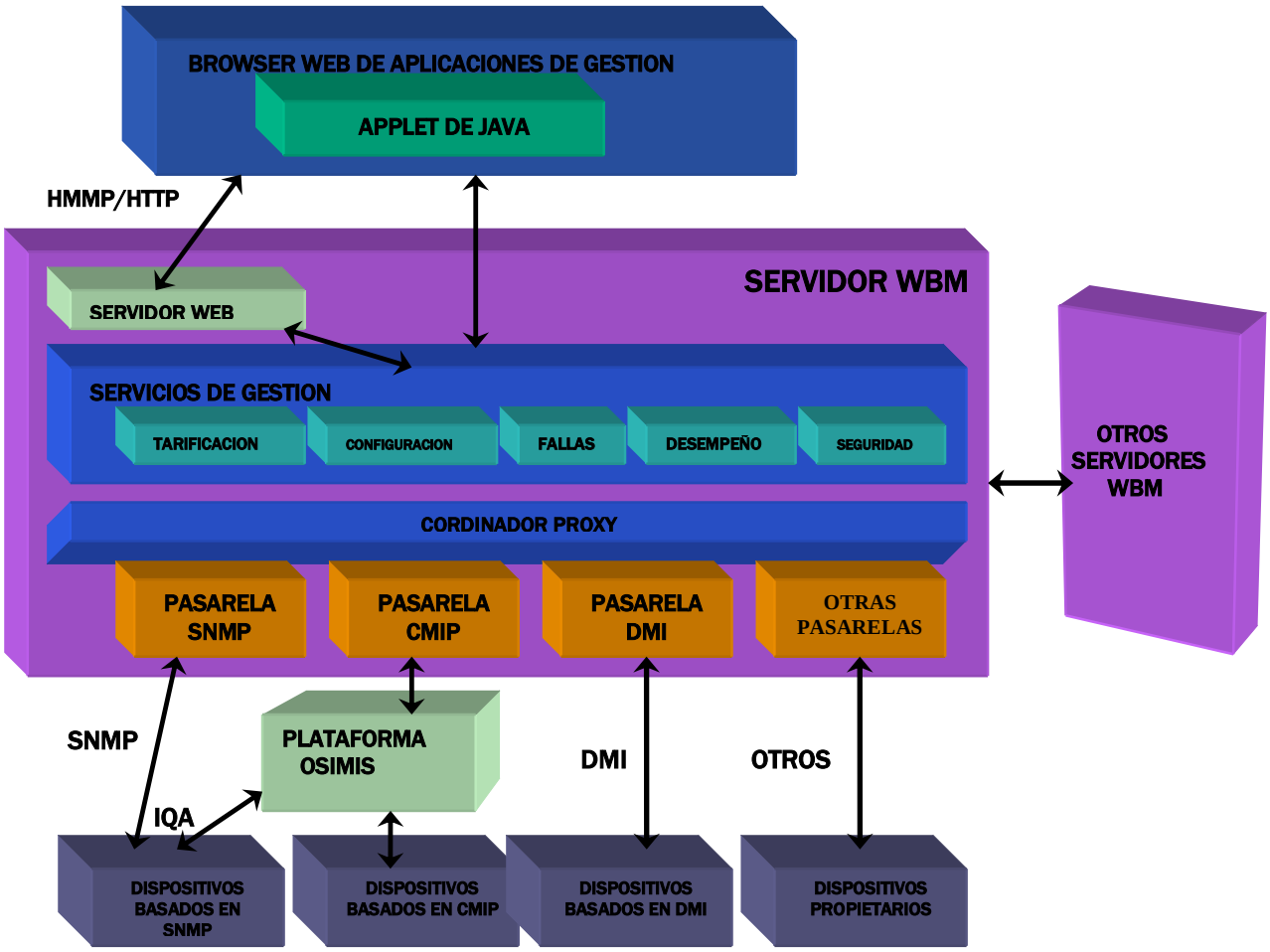


Figura #1. Plataforma GETWeb

OSIMIS es una plataforma de gestión orientada a objetos, basada en el modelo OSI e implementada principalmente en C++. Esta provee un ambiente para el desarrollo de aplicaciones de gestión, el cual oculta los detalles de los servicios de gestión subyacentes a través de interfaces de programas de aplicación (APIs) orientadas a objetos que facilita a los diseñadores e implementadores concentrarse más en la inteligencia que necesita ser construida dentro de la aplicación de gestión y pasar por alto los mecanismos de acceso al servicio/protocolo de gestión.

OSIMIS fue diseñado desde el comienzo con la intención de soportar la integración de sistemas existentes con facilidades de gestión propietarias o modelos de gestión diferentes, por ello, diferentes métodos para la interacción con recursos gestionados reales son soportados, extendiéndose a recursos débilmente acoplados como es el caso de agentes subordinados y jerarquías de gestión . Se sabe que el modelo OSI es el modelo de gestión básico, que facilita la integración de otros modelos, como es el caso de el SNMP y las tecnologías OMG CORBA.

Realmente OSIMIS no intenta proveer un conjunto de facilidades de gestión OSI, mas bien pretende mostrar como puede ser explotada la riqueza de las facilidades de gestión OSI y servir como una plataforma de gestión OSI genérica, en particular las APIs, por ejemplo ayudan a ocultar los detalles del acceso a la información de gestión a través de CMIS/P.

OSIMIS ha sido desarrollado usando el ambiente de desarrollo ISO (ISODE [11]) y dispone de partes genéricas y específicas. Las partes genéricas son infraestructura orientada a objetos para el desarrollo de agentes y gestores OSI y un conjunto de aplicaciones de gestión genérica por ejemplo bases de información de gestión especificas o independientes. Las partes especificas son agentes con MIBs [12](Management Information Base) especificas y sus aplicaciones de gestión asociadas. Se debe de notar que el énfasis de OSIMIS esta en proveer una infraestructura genérica mas que aplicaciones especificas.

OSIMIS fue desarrollado en la University College of London (UCL) como un producto envuelto en diferentes proyectos de investigación Europea. Desafortunadamente en la actualidad no hay soporte del producto por parte de la UCL o de sus desarrolladores.
La plataforma a sido desarrollada como parte de los siguientes proyectos de investigación Europea:

- ESPRIT I INCA(Integrated Network Communications Architecture): Este tuvo como objetivo el diseño y desarrollo de una infraestructura y aplicaciones OSI. Desarrolló algunas partes de ISODE, el servicio de directorio QUIPU [13] y la primera versión de OSIMIS que incluía una versión inicial de CMIS/P y agentes/gestores específicos para la gestión del protocolo de transportes ISO que fue desarrollado dentro de ISODE.
- ESPRIT II PROOF (Primary Rate ISDN OSI Office Facilities): Este se encargo de investigar, diseñar y desarrollar el IP/X.25 para una tasa primaria de pasarelas ISDN y herramientas de gestión OSI asociadas. Desarrolló la primera versión del sistema de gestión genérico GMS (Generic Management System) de OSIMIS, adicionando luego el control Log y la MIB OSI de Internet con sus agentes y gestores que permitían gestionar aquellas pasarelas desarrolladas.
- RACE I NEMESYS (Network Management using Expert Systems): Este tuvo el objetivo de investigar el tráfico y la calidad del servicio de gestión para integrarlo con comunicaciones de banda ancha y continuó con el desarrollo del GMS.
- RACE II ICM (Integrated Communications Management): Este trabajo fue la continuación del proyecto NEMESYS sobre una infraestructura IBCN real, no simulada.
- ESPRIT III MIDAS (Management In a Distributed Application/Service environment): Este trabajo se encargo de demostrar que el modelo de gestión OSI es igualmente adaptable a la gestión de aplicaciones distribuidas tales como el directorio X.500 y el servicio Mail X.400.

Todos estos proyectos pueden ser encontrados y estudiados en profundidad en la siguiente dirección de Internet: http://www.cordis.lu/esprit/home.html.

La plataforma OSIMIS es de dominio público y su última versión OSIMIS 4.0 que fue la usada para el desarrollo del trabajo de tesis, puede ser obtenida libremente

para uso no comercial por medio de Internet desde la URL: www.cs.ucl.ac.uk/research/osimis/index.htm, para lo cual es necesario registrarse, obteniendo así un número de registro que será utilizado para la desencriptación de los archivos; desde esta misma dirección se puede obtener la versión 8.0 de ISODE, la cual es necesaria para el correcto funcionamiento de OSIMIS como se podrá observar en los capítulos siguientes.

La plataforma OSIMIS cumple con el estándar TMN, y por tanto ha de regirse por el modelo OSI, el cual utiliza un lenguaje especifico para declarar y definir la información y un protocolo especifico para transportar y acceder de manera segura y adecuada esta información; el numeral siguiente nos permitirá observar como esta definido el modelo de información utilizado por OSI aclarándonos estos conceptos.

1.2 EL MODELO DE INFORMACION DE GESTION OSI

El modelo de gestión de red OSI se basa sobre el paradigma orientado a objetos y de esta forma la gestión de recursos reales físicos o lógicos se hace posible a través de abstracciones de estos conocidos como objetos gestionados MOs. Las clases de objetos gestionados MOCs (Management Objects Class) asociadas a varios recursos de comunicación han sido especificadas por grupos de estandarización en un lenguaje formal conocido como GDMO [14] (Guidelines Definition Management Objects). El esquema de información de gestión, su herencia y jerarquía de contenimiento junto con los atributos gestionados, operaciones, acciones, notificaciones y comportamiento de cada clase de objeto gestionado son formalmente especificadas en este lenguaje. Estas clases de objetos gestionados son organizadas en una herencia jerárquica mientras los objetos gestionados que componen una instancia MIB en un sistema gestionado son organizados en el MIT[15] (Management Information Tree), de acuerdo a las relaciones de contenimiento. En términos más simples, una relación de herencia

especifíca que un objeto es de un tipo de otro objeto, mientras el contenimiento especifíca que un objeto es parte de otro objeto.

En el esquema MIT, las relaciones de contenimiento entre clases son especificadas por los nombres enlazados (“Name Bindings”) definidos dentro de la MIB que las especifíca. Cada clase de objeto tiene un atributo que es usado dentro de la definición del nombre enlazado dentro de la plantilla GDMO, el cual junto con su valor identifican al objeto unívocamente dentro del alcance de la clase que lo contiene. Esto es conocido como RDN (Relative Distinguished Name), y la secuencia de todos los RDNs desde el top del árbol a un objeto gestionado constituye su DN (Distinguished Name), el cual es único dentro del alcance del sistema gestionado. Un ejemplo de una instancia MIT y su esquema son mostrados en la Figura #2.

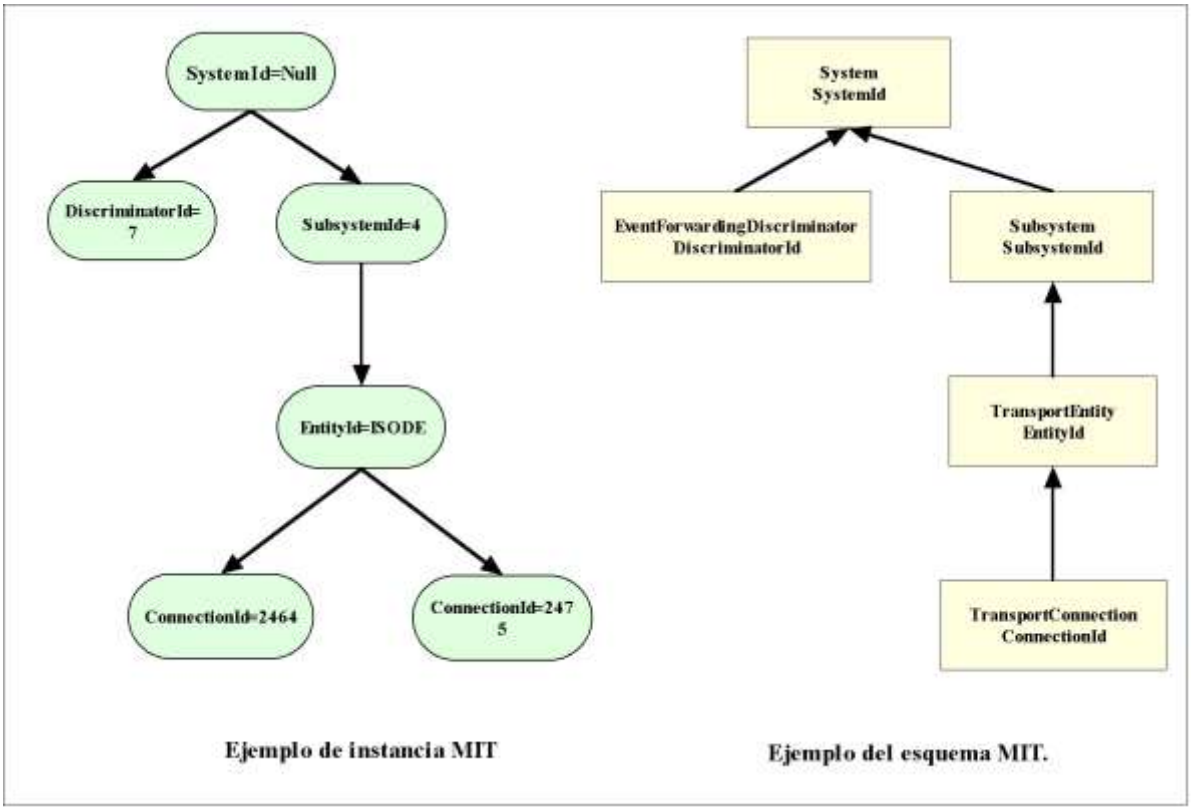


Figura #2. Esquema e Instancia MIT

Una instancia de objeto gestionado puede pertenecer a muchas clases alomorfas, permitiendo ésta ser gestionada como cualquiera de ellas. Este es un concepto muy importante ya que permite extender una clase de objeto gestionado de acuerdo a necesidades especificas sin impedir a otros sistemas gestionar ésta. Todas las clases de objeto gestionado son derivadas desde la clase genérica Top, la cual tiene como sus atributos la clase actual, la lista alomorfa de clases, los nombres enlazados y la lista de paquetes en la instancia de objeto. Aclaremos que un paquete es una colección de atributos, notificaciones, operaciones y comportamiento, y ellos pueden ser condicionales, incrementando aun más la flexibilidad de gestión.

CMIS/P posibilita el acceso a la información de gestión, proveyendo un rico conjunto de operaciones las cuales junto con las especificaciones de clases de objetos gestionados identifica unívocamente la interfaz de gestión entre un gestor y un sistema gestionado. Estos servicios son conocidos como:

- **M-GET:** el cual lee atributos de gestión
- **M-SET:** el cual modifica atributos de gestión
- **M-ACTION:** el cual desempeña una acción sobre un objeto gestionado
- **M-CREATE:** el cual crea un objeto gestionado
- **M-DELETE:** el cual borra un objeto gestionado
- **M-EVENT-REPORT:** el cual posibilita a un sistema gestionado notificar a una aplicación de gestión que un evento a ocurrido.

Todas estas operaciones tienen como parámetros comunes el nombre distinguido y la clase de objeto gestionado. El nombre distinguido permite a un objeto particular ser direccionado mientras la clase de objeto permite ser gestionada como una clase alomorfa particular. Las operaciones M-GET, M-SET, M-ACTION y M-DELETE pueden ser aplicadas a más de un objeto usando el scoping, el cual permite direccionar objetos subordinados de un nivel particular, bajo un nivel

particular o el subarbol completo. La selección de objetos puede ser controlada por parámetros de un filtro, que contendrá afirmaciones sobre valores de atributos conectados por operadores lógicos.

CAPITULO II

CONOCIENDO ISODE Y OSIMIS

En este Capitulo empezamos el estudio de la plataforma de gestión, presentando su estructura y que partes la conforman; es indispensable antes de continuar, aclarar que para lograr un buen entendimiento de todo el trabajo de grado se deben poseer conocimientos en TMN, el modelo OSI, la programación orientada a objetos (más específicamente con C++), los protocolos CMIS/P, la gestión SNMP, los lenguajes de especificación GDMO y ASN.1, y del sistema operativo UNIX.

2.1 PLATAFORMA ISODE

ISODE es una plataforma para el desarrollo de servicios OSI y sistemas distribuidos, que provee las capas superiores de la pila OSI siguiendo las recomendaciones ISO/ITU-T e incluye herramientas para manipulación ASN.1 permitiendo la generación de llamadas a operaciones remotas. ISODE provee dos servicios de OSI: El Servicio de Directorio (X.500 [16]) y la Transferencia de Archivos FTAM[17] (File Transfer Access and Management).

ISODE está implementado en el lenguaje de programación C y se ejecuta en la mayoría de las versiones del sistema operativo UNIX. ISODE no provee los protocolos de las capas inferiores de OSI como por ejemplo X.25 o CLNP (Connectionless Network Protocol), pero posee una implementación para estaciones de trabajo basadas en UNÍX las cuales son accesibles a través de la

interfaz del núcleo (kernel). Las capas superiores del modelo OSI implementadas por ISODE son: transporte, sesión y presentación.
Además ISODE provee los elementos de servicio de control de asociación, operaciones remotas y transferencia confiable; ACSE (Asociation Control of Service Element), ROSE (Remote Operations of Service Element), y RTSE (Reliable Tranfer of Service Element) respectivamente. Los cuales al ser usados en conjunto con el soporte ASN.1 actúan como un bloque constructor para los servicios de niveles más altos. La pila ISODE es por lo tanto un conjunto de librerías enlazadas con aplicaciones que usan estos servicios.

La manipulación de ASN.1 es muy importante para las aplicaciones distribuidas OSI. El acercamiento de ISODE para una Interfaz de programación descansa sobre la abstracción fundamental conocida como elemento de presentación (PE), esta es una estructura en C genérica capaz de describir cualquier tipo de dato ASN.1 y el compilador de ASN.1 en lenguaje C conocido como pepsy, es el encargado de producir las representaciones correspondientes de la sintaxis. Uno de los más importantes conceptos pioneros en ISODE es el de poder trabajar sobre diferentes pilas de protocolos de las capas inferiores, lo cual se logra a través del puente del servicio de transporte (TS-bridge).

ISODE provee una implementación del protocolo de transporte ISO (TP) clase 0 sobre X.25, o también sobre el Internet usando RFC1006 [18], en el cual TCP es tratado como un servicio de red confiable. El protocolo de sesión ISODE puede correr también sobre ISO-TP clase 4 y el protocolo de red no orientado a conexión (CLNP). El puente del servicio de transporte puede ser usado para enlazar las sub-redes de diferentes comunidades y proveer una interoperabilidad end-to-end ocultando la heterogeneidad de la tecnología de las redes subyacentes, ya que las combinaciones antes mencionadas constituyen la vasta mayoría de redes desplegadas actualmente. La Figura#3, aclara los conceptos anteriormente nombrados.

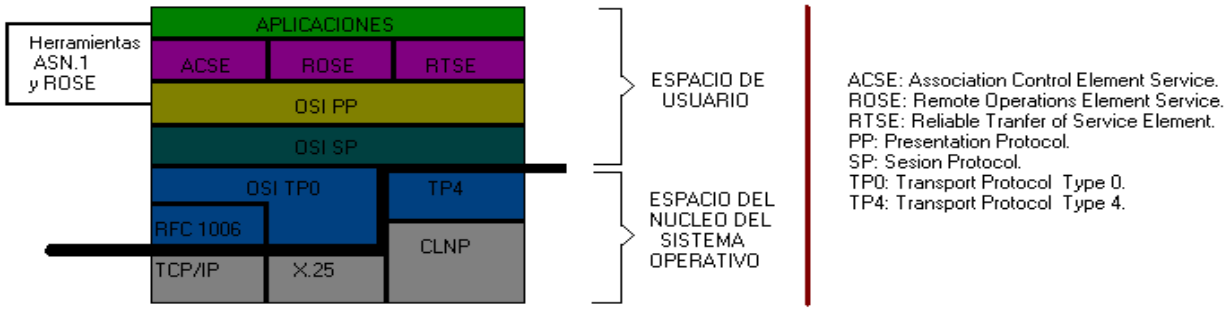


Figura #3. Pila ISO de ISODE

A continuación se muestra el ambiente de desarrollo proporcionado por ISODE:

2.1.1 PROTOCOLOS Y SERVICIOS:

- El protocolo de transporte OSI clase 0, el cual puede correr sobre X.25 o igual sobre TCP/IP, tratando el último como servicio de red confiable, usando el método RFC1006, también se puede ejecutar protocolos de alto nivel sobre el OSI TP clase 4 en el servicio CLNP.
- Los protocolos de sesión y presentación de OSI, incluyendo una versión ligera del último que puede correr directamente en TCP usando el método RFC1085 [19].
- El servicio de control de asociación OSI (ACS-Asociation Control Service), el servicio de operaciones remotas (ROS – Remote Operation Service), y el servicio de transferencia confiable (RTS- Reliable Transfer Service), los cuales son bloques fundamentales de construcción para protocolos de alto nivel y servicios.
- El servicio de gestión y acceso a la transferencia de archivos OSI (FTAM), el cual usa ACS y RTS
- El servicio de acceso a directorio OSI (DSA – Directory Service Access), el cual usa ACS y ROS.

2.1.2 HERRAMIENTAS INCLUIDAS EN ISODE:

- Un compilador para el lenguaje ASN.1 OSI, basado en el lenguaje de programación C; en el cual la estructura de datos producida es diferente para el API de XOM X/Open y para el compilador ISODE (OSIMIS puede utilizar el estándar XOM X/Open [20]como una alternativa a ISODE).
- Un compilador para la plantilla de operaciones remotas que sirve como un generador de stubs para estas.

2.1.3 APLICACIONES:

- Un servicio FTAM para el sistema operativo UNIX.
- Un agente de servicios genéricos de directorio (DSA), que es soportado por una clase de objetos directorio y un conjunto de agentes de usuario del directorio (DUAs). Las interfaces de usuario están basadas en terminales básicos y gráficos; el DSA permite el direccionamiento de aplicaciones en OSI y el servicio de ubicación transparente de OSIMIS.
- Un puente para el servicio de transporte (TS), el cual permite la interoperabilidad de aplicaciones sobre pilas OSI incluyendo diferentes capas bajas, a saber TPO sobre X.25, TPO sobre RFC 1006 y con TCP/IP y TP4 sobre CLNP.

2.2 PLATAFORMA OSIMIS

OSIMIS utiliza ISODE como una pila subyacente de protocolos de capas superiores OSI, aunque también soporta al XOM/XMP como una alternativa. Utilizar ISODE proporciona la ventaja de permitir a OSIMIS soportar las diferentes tecnologías de red antes nombradas como X.25, lo que es esencial para un ambiente de gestión distribuida compleja.

Los servicios y la arquitectura de OSIMIS es mostrada en la Figura #4. En la parte correspondiente a las capas contenidas sobre y bajo la línea roja, son

aplicaciones y programas mientras que el resto, los de la parte baja y de ambos lados, son bloques de construcción realizados como librerías. La parte más baja muestra las aplicaciones genéricas proveídas; de estás, las herramientas ASN.1 y GDMO son esenciales para el soporte “off-line” en la realización de nuevas MIB´s.

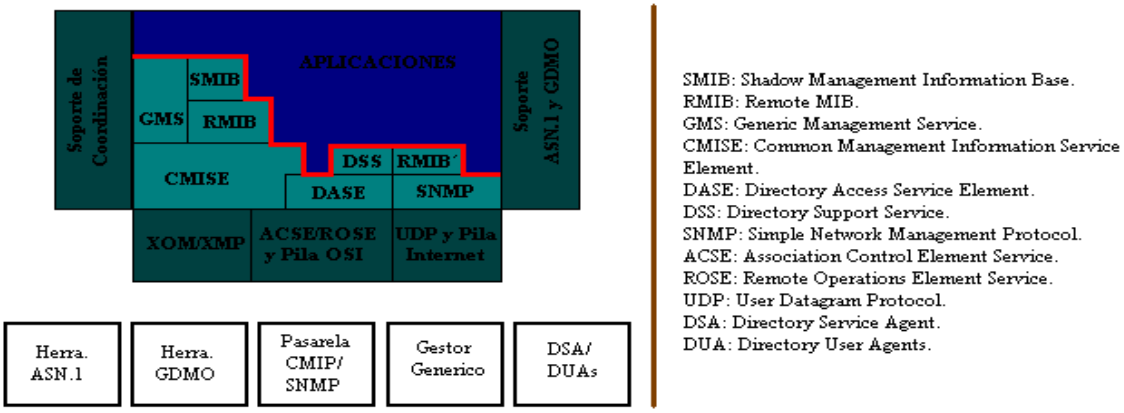


Figura #4. Arquitectura en capas de OSIMIS

La línea roja indica todas las API´s que una aplicación puede usar. En la práctica la mayoría de las veces las aplicaciones usan solamente el sistema de gestión genérico (GMS) y el API de MIB remoto (RMIB – Remote MIB), sumados a las API´s de soporte de coordinación y ASN.1 de alto nivel.

OSIMIS como se ha dicho está soportado sobre ISODE, de esta manera en la anterior arquitectura los servicios están subdivididos en servicios de la capa baja (soportados por ISODE) y servicios de la capa alta (facilitados por OSIMIS), la pila OSI sobre el nivel ACSE/ROSE es proveída por ISODE, mientras la pila Internet sobre el UDP es proveída por el kernel del sistema operativo UNIX. Sobre estos OSIMIS provee realizaciones procedimentales (no orientadas a objetos) de CMIS/P y SNMP.

El GMS y las APIs RMIB y SMIB (Shadow MIB) logran hacer más amistoso al usuario el trabajo con CMIS, mientras que el ASN.1 es ocultado, para que sea mas amistoso, tras una API de sintaxis abstracta de alto nivel la cual encapsula el PE de ISODE; Existe una infraestructura similar a la RMIB para facilitar el acceso a objetos SNMP conocida como RMIB´, y finalmente el DSS provee acceso al directorio X.500 para resolver direcciones, ubicaciones y otros servicios.

Lo anterior puede ser observado en las Figuras #5 y #6.

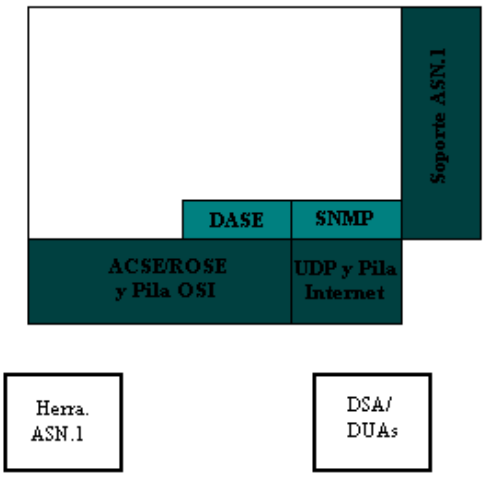


Figura #5. Componentes Proveídos por ISODE

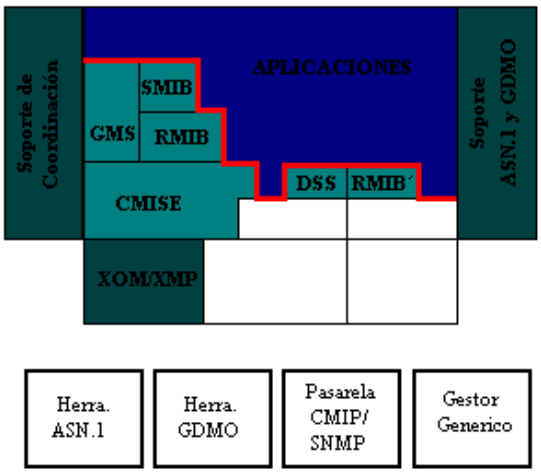


Figura #6. Componentes Proveídos por OSIMIS

En la Tabla #1 describiremos los servicios que puede prestar la plataforma OSIMIS y de los cuales nosotros no haremos uso por presentar problemas en su funcionamiento o por no ser de nuestro interés:

SERVICIO	RAZON PARA NO USARLO
Un servicio de soporte de presentación, el cual permite el uso de diferentes tecnologías para interfaces de usuario actualmente sobre Motif, InterViews y TCL/TK	Este servicio no se coloco en funcionamiento debido a que en la instalación de OSIMIS utilizamos un sistema operativo diferente a aquellos con los cuales son compatibles estas tecnologías (ver el apartado 2.4); el porque la decisión de utilizar un sistema operativo diferente lo aclararemos en el capitulo III.
Un servicio que Permite obtener un resumen de objetos en la gestión del sistema el cual puede reportar arbitrariamente y periódicamente valores elegidos, posiblemente a manera de buffer y después facilitar estadísticas.	Este servicio no se logro hacer funcionar correctamente, pues siempre bloqueaba el agente de gestión, parece ser un "*bug*" de programación o total incompatibilidad con los compiladores y sistema operativo usados.
Un mecanismo de ubicación transparente usando la implementación ISODE del servicio de directorio OSI	Para su funcionamiento este se basa en el servicio X.500 que presta ISODE, para el cual no logramos crear adecuadamente un DSA (Directory Service Agents) local, que nos permitiera probar este servicio.
Soporte para la inclusión de mecanismos de seguridad para autenticación.	Los mecanismos de seguridad y autenticación se basan en el servicio X.500 por la misma razón anterior no logramos usar este servicio.
Una API con soporte para el estándar industrial X/Open, XOM/XMP	Esta es una opción para utilizar OSIMIS sobre otra plataforma distinta a ISODE, lo cual no fue de nuestro interés.

Tabla #1. Servicios no usados de OSIMIS.

Es necesario aclarar que al no ser usados estos servicios, OSIMIS no pierde potencial como plataforma de gestión, y no limita nuestros objetivos en el trabajo de grado.

Ahora enumeraremos los servicios y aplicaciones proveídos por OSIMIS, y puestos en funcionamiento en este proyecto:

1. Una implementación completa del CMIS/P de OSI, el cual usa el ACS, ROS de ISODE y soporta el ASN.1.

2. Un mecanismo de coordinación que usa la infraestructura orientada a objetos y que permite organizar una aplicación de gestión compleja como

una clase completamente manejada por eventos; esto permite fácilmente la intercomunicación con otros mecanismos similares.

3. Soporte ASN.1 de alto nivel, el cual oculta completamente los mecanismos de codificación y decodificación a través del encapsulamiento de objetos en C++.

4. El GMS, que es un sistema orientado a objetos el cual cuenta con unas API´s que permiten generar nuevas clases de objetos que actúan como agentes OSI. Además soporta los mecanismos de coordinación, la gestión de objetos, el reporte de eventos y facilidades de control Log.

5. Un compilador para el lenguaje de especificación de objetos gestionados GDMO OSI, el cual complementa el GMS y produce el soporte para la gestión de objetos en tiempo de ejecución.

6. Un gestor remoto de la MIB con acceso a la API de alto nivel, el cual hace más amistoso CMIS y ASN.1, sin sacrificar funcionalidad o desempeño.

7. Una pasarela genérica entre CMIS/P y SNMP, la cual es manejada por un traductor genérico entre Internet y MIB's OSI.

8. Soporte de un monitor métrico de objetos gestionados, el cual habilita contadores e indicadores para monitorear los posibles valores producidos, incluyendo umbrales y marcas soportando estadísticas.

9. Un completo estándar para reporte de eventos y funciones de gestión de sistemas “logging” y soporte para los objetos, gestión de estados y reporte de alarmas.

10. Un compilador ASN.1 orientado a objetos el cual produce objetos C++ correspondientes a una sintaxis ASN.1 arbitraria.

11. Un conjunto de aplicaciones de gestión genérica que permiten realizar operaciones de gestión con el completo poder de CMIS.

Con los servicios anteriores tenemos el soporte para realizar una meta-gestión si lo deseamos, es decir podremos gestionar al propio sistema de gestión.

Todos estos servicios y aplicaciones serán estudiados a fondo en el Capitulo IV.

2.3 AMBIENTES DE OPERACIÓN E INTEROPERABILIDAD

Es conocido que OSIMIS opera en las siguientes variantes del sistema operativo UNIX con los compiladores de C y C++, Como se indica en la Tabla #2.

Sistema Operativo	Compilador
Solaris 2.x	Sun C/C++ 4.1 GNU C/C++ 2.6.3
SunOS 4.1.3	GNU C/C++ 2.6.3
Linux 1.2.x	GNU C/C++ 2.6.3
HP-UX 9.x	GNU C/C++ 2.5.4

Tabla #2. Sistemas Operativos en los que trabaja OSIMIS

La compatibilidad hacia atrás del GNU 2.5.8 es garantizada en Solaris, SunOs, Linux. OSIMIS usa el ISODE 8.0 en la capa superior de la pila OSI sobre todas las plataformas o el Sun XOMP/XMP versión 4.1 en solaris 2.x. Las versiones Tcl/Tk soportadas son Tcl 7.4 y Tk 4.0. En el próximo capitulo se detallara en que sistemas se intento instalar durante el trabajo de grado y en cual se logró con éxito.

Se sabe que OSIMIS interopera con IBM NetView TMN, HP OpenView, Bull ISM, Ericsson TMOS e. igualmente con la pila CMIP de Retrix.

CAPITULO III

INSTALACION DE ISODE Y OSIMIS

Este capítulo contiene los pormenores que explican cada paso necesario para conseguir la compilación, instalación y puesta a punto de las plataformas ISODE y OSIMIS, permitiéndonos de esta manera mostrar como se pudieron superar ciertos problemas acaecidos, aunque es importante resaltar algo ya dicho, existen problemas no solucionados referentes a ciertos recursos que podría prestar la plataforma y que no logramos colocar en correcto funcionamiento, como son el servicio de directorio (X.500) y la interfaz de usuario gráfica, pero igualmente debemos resaltar que el objetivo propuesto para el trabajo de grado se llevo a cabo, puesto que los recursos y todos los servicios necesarios para realizar la gestión usando la plataforma OSIMIS funcionan correctamente.

En la Tabla #2 del Capitulo II sección 2.3 presentamos los sistemas operativos y compiladores para los cuales los desarrolladores de la plataforma aseguran compatibilidad, como podemos observar, las versiones de ambos sistemas operativos y compiladores son bastante viejas, lo cual es lógico puesto que la ultima versión de OSIMIS en la que los desarrolladores trabajaron fue liberada en 1994. Desde el principio en el trabajo de grado la intención fue lograr instalar la plataforma en un sistema operativo accesible para los estudiantes de la universidad y de libre distribución para que ellos puedan estudiar y realizar practicas de gestión, por tal razón concentramos nuestros esfuerzos en la correcta

instalación de la plataforma sobre el sistema operativo Linux, lo cual logramos luego de realizar muy variados intentos y superar diferentes inconvenientes que serán presentados posteriormente.

Las siguientes secciones de este capítulo explicaran los pasos necesarios para compilar, configurar e instalar tanto ISODE como OSIMIS detallando los cambios y acciones necesarias a seguir, con la intención de ser un manual para el lector de la monografía y que de esta manera pueda colocar fácilmente en funcionamiento la plataforma OSIMIS.

3.1 VISION GENERAL DE LA INSTALACION.

La Figura #7 presentada a continuación nos permitirá tener una visión global de los pasos a seguir para cumplir con el proceso de instalación adecuadamente, estos pasos serán estudiados a fondo y desglosados en los numerales siguientes.

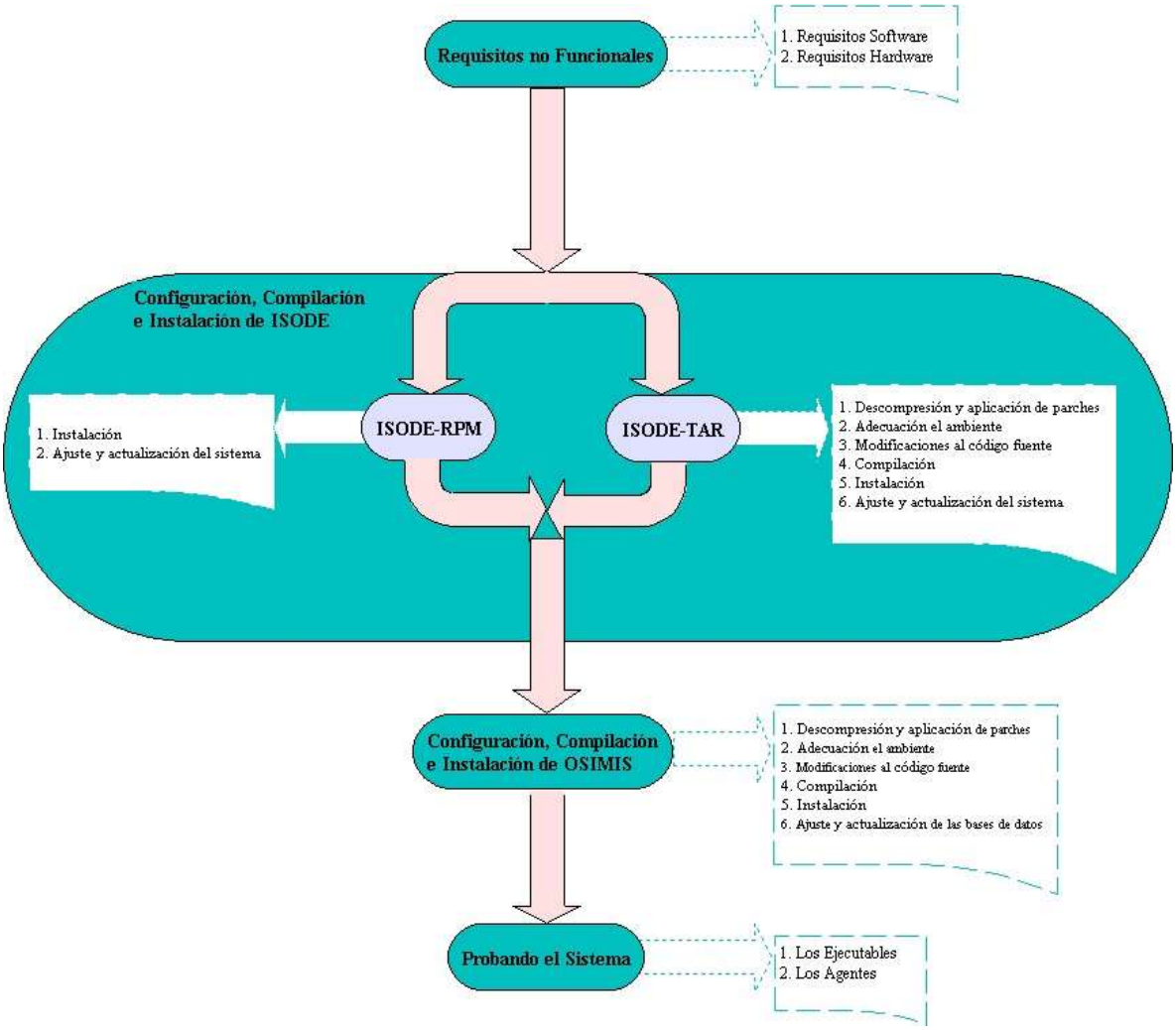


Figura #7 Visión General de la Instalación

3.2 REQUISITOS NO FUNCIONALES.

La instalación de la plataforma OSIMIS requiere el cumplimiento de ciertos requisitos hardware y software, asegurar el cumplimiento de tales requisitos debe ser por tanto nuestro primer paso para lograr la instalación, a continuación presentamos estos requisitos a cumplir:

3.2.1 REQUISITOS HARDWARE.

La plataforma a sido instalada con éxito en sistemas con hardware diferente, la Tabla #3 muestra la lista de sistemas en que la hemos probado y por tanto para los cuales podemos asegurar compatibilidad.

Procesador	Velocidad del Procesador	Espacio Minimo en Disco Duro.	Memoria RAM
Pentium II	500 MHz	800 Mb	64 Kb
AMD K6 II	500 MHz	800 Mb	64 Kb
Pentium III	700 MHz	800 Mb	128 Kb
AMD K6 II	450 MHz	800 Mb	128 Kb

Tabla #3 Requisitos Hardware.

3.2.2 REQUISITOS SOFTWARE.

Luego de muchos intentos usando diferentes versiones de sistemas operativos tanto Linux como Sun para instalar la plataforma, logramos hacerlo sólo en la versión “Linux MX - Basado en Red Hat/Intel 5.1 (Liberación Manhattan)”, por tanto este sistema operativo es el requisito software base para la instalación de la plataforma; la Tabla #4 muestra las librerías y compiladores que son obligatorios para poder instalar la plataforma, y los cuales son ofrecidos por el sistema operativo antes mencionado.

Linux Red-Hat 5.1 (Manhattan) con kernel 2.0.35	
Librerias	**Compiladores**
• libutil.so.1 • libm.so.6 • libcrypt.so.1 • libc.so.6 • ld-linux.so.2	• gcc 2.7.2.3 • gawk 3.0

Tabla #4 Requisitos Software.

3.3 CONFIGURACION, COMPILACION E INSTALACION DE ISODE.

La instalación de la plataforma ISODE se puede realizar tomando dos caminos diferentes, estos son, usando los archivos fuente (el cual denotaremos como ISODE-TAR) o usando el paquete en formato RPM [21] (el cual denotaremos como ISODE-RPM), cualquiera de los dos caminos nos conducirá a la correcta instalación de ISODE tal como es necesario para el funcionamiento de OSIMIS, obviamente es mucho más sencilla la instalación de ISODE utilizando el paquete en formato RPM, pero para nosotros fue necesaria la instalación de ISODE usando los archivos fuente para poder realizar estudios y pruebas que nos llevaron a la corrección de problemas en la comunicación con los sockets, esto será tratado y explicado en el numeral 3.4.3 en la parte correspondiente a las modificaciones en el código fuente de OSIMIS, por lo tanto consideramos de gran importancia explicar esta forma de instalación de ISODE para permitir posteriores estudios en busca de lograr el funcionamiento de servicios como el X.500.

A continuación explicaremos entonces las dos formas de compilación, instalación y ejecución que usamos para ISODE.

3.3.1 ISODE-TAR.

3.3.1.1 Descompresión y Aplicación de Parches.

Al tener correctamente instalado el sistema operativo Linux Red-Hat 5.1 podemos empezar con la instalación de ISODE, primero que todo es necesario tener privilegios de súper-usuario a fin de poder instalar el software, aunque estos no son necesarios para la compilación y configuración. La distribución que usamos de ISODE es la versión 8.0 la cual se encuentra en el CD que acompaña la monografía o puede ser obtenida desde la University College of London en la URL http://www.cs.ucl.ac.uk/research/osimis la cual se encuentra comprimida, en formato TAR y acompañada de varios parches como podemos observar en la Tabla #5.

	Archivos	**Tamaño**
Archivo Principal	isode-8.tar.Z	5339 Kb
Parche #1	isode-8-patch1.tar.Z	11Kb
Parche #2	isode-8-patch2-solaris.tar.Z	231 Kb
Parche #3	isode-8-patch3-linux.tar.Z	121 Kb
Parche #4	isode-8-patch4-linux.tar.	18Kb

Tabla # 5 Archivos comprimidos de ISODE

la descompresión y aplicación de parches la realizamos de la siguiente forma:

- ubicados en el directorio raíz creamos un directorio, para nuestro caso lo llamamos tesis (***mkdir tesis***)
- dentro de este directorio copiamos los archivos comprimidos de ISODE para descomprimirlos y aplicar los parches, lo cual realizamos en tres pasos, es necesario aclarar que el parche #2 para solaris no debemos aplicarlo.
    1. Descomprimimos el archivo principal con "***zcat isode-8.tar.z | tar xvf –***", al usar esta instrucción para descomprimirlo se creará el directorio "isode-8.0" dentro del cual estarán los archivos fuente. Desde ahora el símbolo "$" lo usaremos para indicar que estamos dentro del directorio isode-8.0 y todo lo que este después del símbolo estará dentro del directorio, es decir /tesis/isode-8.0=$.
    2. Debemos descomprimir el parche #1 usando "***zcat isode-8.patch1.tar.z | tar xvf –***" el cual creara un directorio llamado ./isode-8-patch1 y dentro de este existe un archivo con el mismo nombre que debemos copiar dentro del directorio isode-8.0 y aplicarlo usando " ***$ patch –p0 < isode-8-patch1***".
    3. Para aplicar los parches #3 y #4 no es necesario utilizar la utilidad patch de Linux, sólo es necesario descomprimirlos sobre el directorio /tesis y ellos reconocerán el directorio ./isode-8.0 y reemplazaran los archivos necesarios, por lo tanto usamos "***zcat isode-8.patch3.tar.z | tar xvf –***" y luego "***zcat isode-8.patch4.tar.z | tar xvf –***", es muy importante respetar el orden aquí dado.

3.3.1.2 Adecuación del Ambiente.

Es necesario configurar y realizar ciertos cambios en algunos archivos para efectuar la compilación con éxito. Para esto debemos entrar al directorio /config (***$ cd config***), dentro de este encontramos una cantidad de bases de datos y de

archivos con extensiones .make y .h para diferentes sistemas operativos, debemos seleccionar aquellos que emparejan con el nuestro, que en este caso son bsd42.make y bsd42.h, los cuales son para un sistema UNIX 4.2BSD genérico sobre el cual se basa nuestro sistema Linux.

Los archivos $ config/CONFIG.make y $ h/config.h y las bases de datos son usados por el propio software de ISODE para configurarse durante el proceso de compilación, por lo tanto es necesario realizarles algunos ajustes y colocarlos en directorios específicos, esto es lo que veremos a continuación.

3.3.1.2.1 Archivo bsd42.make: En este archivo es necesario primero editar las siguientes variables:

BINDIR :	Donde se instalan los programas de usuario
SBINDIR:	Donde se instalan los programas del administrador
ETCDIR:	Donde se instalan los archivos del administrador
LOGDIR:	Donde se guardan los archivos Log
INCDIR:	Donde se instalan los archivos include
LIBDIR:	Donde se instalan las librerías de objeto
LINTDIR:	Donde se instalan las librerías Lint
SYSTEM:	Indica como crear las librerías loader
MANDIR:	Donde se instalan las paginas de ayuda
MANOPTS:	Indica en que formato están las ayudas.

Con los valores que se indican en la Figura #8. Debemos también asegurar que existan los directorios usados, y luego renombramos este archivo como CONFIG.make sobrescribiéndolo así: ***"$ cp config/bsd42.make config/CONFIG.make "***. Estas rutas que se definen serán usadas en el momento de la instalación del software y aquí colocamos las usadas por nosotros, el lector podrá seleccionarlas a su gusto.

```
...⇒⇒⇒⇒⇒⇒ Línea # 28
#############################################################################
# Options
#############################################################################

OPTIONS         =       -I. -I$(TOPDIR)h $(PEPYPATH) $(KRBOPT)

HDIR        =   $(TOPDIR)h/
UTILDIR=        $(TOPDIR)util/
BINDIR  =       /usr/bin/
SBINDIR         =       /usr/sbin/
ETCDIR =        /etc/
LOGDIR=         /usr/isodelogs/
INCDIR  =       /usr/include/isode
INCDIRM         =       $(INCDIR)/
PEPYDIRM    =   $(INCDIR)pepy
PEPYDIR         =       $(PEPYDIRM)/
PEPSYDIRM   =   $(INCDIR)pepsy
PEPSYDIR    =   $(PEPSYDIRM)/
LIBDIR  =       /usr/lib/
LINTDIR=        /usr/lib/lint/

LIBISODE    =   $(TOPDIR)libisode.a
LIBDSAP         =       $(TOPDIR)libdsap.a

SYSTEM          =       -bsd42
MANDIR          =       /usr/man/
MANOPTS         =       -bsd42


#############################################################################
# Shared libraries
#############################################################################



. . .
```

Figura #8 Edición del Archivo bsd42.make

3.3.1.2.2 Archivo bsd42.h: Este archivo sólo necesita ser copiado dentro de la carpeta "$ /h"sin necesidad de editarlo, pero renombrándolo como config.h, esto lo hacemos así: **" *$ cp config/bsd42.h h/config.h* "**.

3.3.1.2.3 Bases de Datos: Ahora nos encontramos con cinco bases de datos diferentes que contienen información de la máquina en la cual estamos trabajando, las cuales son:

aliases.local: Archivo alias de ISODE, especifica el mapeo entre cadenas amistosas y nombres distinguidos.

services.local: Base de datos de servicios de ISODE, mapea entre servicios, programas y selectores.

entities.local: Contiene las plantillas para servicios locales o específicos como NSAP[22], TSAP[23].

macros.local: Permite la definición de Macros.

objects.local: Permite el mapeo entre descriptores de objeto e identificadores de objeto.

Estos archivos están dentro del directorio "$ config/", y se pueden editar si hay necesidad de adicionar algún servicio, programa, etc. Pero en el sistema operativo sobre el cual trabajamos no es necesario realizarles cambios, así que solo los copiaremos al directorio adecuado para que los pueda usar ISODE en el momento de la compilación, esto lo hacemos usando **" *$ cp config/*.local support/* ".**

3.3.1.3 Modificaciones al Código Fuente.

Para lograr la correcta compilación y funcionamiento de la plataforma ISODE es necesario realizar algunos cambios en archivos del código fuente, estos cambios serán expuestos a continuación.

3.3.1.3.1 Archivo general.h.

3.3.1.3.1.1 Descripción: Este archivo es el encargado de definir las compatibilidades generales con el sistema, es decir define la configuración especifica al sistema Linux que estamos usando para que ISODE reconozca que librerías podrá usar, ya que ISODE al igual que OSIMIS permite ser instalado sobre diferentes sistemas operativos como Sun, Solaris y HP-UX y el archivo general.h se encarga de definir los objetivos específicos para el sistema correcto.

3.3.1.3.1.2 Ubicación: El archivo general.h se encuentra en la ruta "/tesis/isode-8.0/h".

3.3.1.3.1.3 Modificaciones: Es necesario acceder a editar el archivo general.h y realizar los siguientes cambios.

- Eliminar la línea donde se redefine la función sprintf() como se muestra en la Figura #9A.

```
...⇒⇒⇒⇒⇒ Línea # 138
#if         defined(BSDSTRS) && !defined(BSD44) && (!defined(BSD43) || defined(SUNOS4) ||
defined(vax) || defined(RT) || (defined(mips) && defined(ultrix))) && !defined(XOS_2)
#if !(defined(__STDC__) && defined(__GNUC__) && defined(mips) && defined(ultrix))
char   *sprintf ();
#endif
#else
int     sprintf ();  → ELIMINAR ESTA LINEA.
#endif

char   *getenv ();

char   *mktemp ();
```

```
. . .
```

Figura #9A Edición del Archivo general.h

- Eliminar la línea donde se redefine la función strdup(), como se muestra en la Figura #9B.

```
...⇒⇒⇒⇒⇒ Línea # 229
void      (*set_smalloc_handler()) ();
char    *smalloc ();
char    *strdup ();       → ELIMINAR ESTA LINEA.

/*    MISC */


char   *sys_errname ();

. . .
```

Figura #9B Edición del Archivo general.h

- Eliminar desde las líneas donde se redefinen la funciones htonl(), htons(), ntohl(), ntohs(), como se muestra en la Figura #9C.

```
...⇒⇒⇒⇒⇒ Línea #265
#ifndef __STDC__
extern time_t time ();
#endif


/*  ntohs etc */

#ifndef   ntohs           →
unsigned short    ntohs ();   →
#endif                  →
#ifndef   htons           →
unsigned short    htons ();   →
#endif                  →    ELIMINAR ESTAS LINEAS.
#ifndef   ntohl           →
unsigned long     ntohl ();   →
#endif                  →
#ifndef   htonl           →
unsigned long     htonl ();   →
#endif                  →



int         char2bcd ();
```

```
int       bcd2char ();
. . .
```

Figura #9C Edición del Archivo general.h

3.3.1.3.1.4 Justificación de las Modificaciones: Todas las modificaciones anteriores son necesarias debido a que cada una de ellas presenta el mismo problema de compilación, este problema lo expresa el compilador con la frase "conflicting types for ..." la cual nos permite conocer que el problema es la redefinición de las funciones, por lo tanto es conveniente utilizar la definición que hacen de ellas los diferentes archivos cabecera del sistema operativo, en lugar de la redefinición que se hace en el archivo general.h, esta redefinición se presenta debido a que estamos utilizando un Kernel del sistema operativo más nuevo (kernel versión 2.0.35) al recomendado para compilar ISODE (kernel versión 1.2.*), y en este kernel nuevo no es necesario la definición de estas funciones en el archivo de compatibilidades generales (general.h).

La Tabla #6 nos muestra que archivos definen estas funciones en nuestro sistema operativo.

Función	Archivo Definiéndola
sprintf()	/usr/include/stdio.h
Strdup()	/usr/include/string.h
htonl(), htons(), ntohl(), ntohs()	/usr/include/netinet/in.h

Tabla #6 Funciones Redefinidas.

3.3.1.3.2 Archivo manifest.h.

3.3.1.3.2.1 Descripción: En este archivo se definen muchas de las constantes que usará el software ISODE, así como también se definen algunas estructuras importantes que dejaremos intactas.

3.3.1.3.2.2 Ubicación: El archivo manifest.h se encuentra en la ruta "/tesis/isode-8.0/h".

3.3.1.3.2.3 Modificaciones: Es necesario acceder al archivo manifest.h y adicionar la línea mostrada en la Figura #10.

```
...⇒⇒⇒⇒⇒ Línea #42
#ifdef    BSD42
#undef    SYS5NLY
#define   BSDSIGS
#define   SIGEMTSIGUSR1    → AGREGAR ESTA LINEA.
#endif


#ifdef    ROS
#undef    SYS5NLY

#define BSDSIGS

. . .
```

Figura #10 Edición del Archivo manifest.h

3.3.1.3.2.4 Justificación de las Modificaciones: Existe un archivo llamado isore.c ubicado en "/tesis/isode-8.0/support/" el cual ayuda a la terminación de los programas TSAP de ISODE (demonio tsapd), el archivo isore.c implementa el monitoreo del tráfico TCP/IP, tan eficientemente como sea posible y envía la señal SIGEMT al demonio tsapd para terminarlo cuando la red genera un evento DATA.INDICATION. El problema que se presenta en la compilación con isore.c es que la señal SIGEMT aparece como no definida, por lo tanto debemos definirla y

la mejor forma de hacerlo es ubicándola en el archivo donde se definen las principales constantes del software ISODE (manifest.h), y debemos darle a esta señal el valor de una señal de “exit”, lo cual podemos hacer dándole el valor de una señal especial de Linux, en este caso usamos SIGUSR1.

3.3.1.4 Compilación.

En este punto ya tenemos todo listo para poder realizar la compilación del software ISODE, y es recomendable resetear los datos de los archivos de configuración del sistema, lo cual hacemos con el siguiente comando:

“ *$ ./make once-only* ”

solo nos queda generar el sistema básico lo cual hacemos de la siguiente manera:

“ *$ ./make everything* “

al ejecutar el comando anterior se desplegaran en la salida estándar (monitor) todas las acciones que esta ejecutando la compilación; en el caso que deseemos enviar estas a un archivo para su posterior estudio podemos escribir:

***“$ ./make everything > ♦♦♦ 2>&1*”**

donde ♦♦♦ es el nombre del archivo en el que deseamos guardar las acciones realizadas.

Con los cambios efectuados anteriormente y siguiendo los pasos dados, el software ISODE debe compilar correctamente y generar los archivos ejecutables necesarios, para asi poder pasar a la instalación de este.

3.3.1.5 Instalación.

La instalación del software base de ISODE es un paso muy corto puesto que en la edición del archivo CONFIG.make (bsd42.make) ya hemos seleccionado los caminos adecuados para los archivos ejecutables, archivos cabecera etc. Por lo tanto ahora sólo debemos escribir:

“ ***$ ./make install*** **“** o

“ ***$ ./make install > ♦♦♦ 2>&1*** **”**

en el ultimo caso ♦♦♦ es el nombre del archivo en el que deseamos guardar las acciones de instalación realizadas.

3.3.1.6 Ajuste y Actualización del sistema.

Para poder empezar a utilizar ISODE y que este trabaje de manera adecuada debemos asegurar la definición de ciertos parámetros en los siguientes archivos del sistema:

3.3.1.6.1 Archivo services: El archivo services en el sistema es el encargado de proveer el mapeo entre nombres amistosos para servicios de Internet y sus tipos de protocolo y número de puerto asignado, cada programa de red debe mirar dentro de este archivo para obtener el número de puerto (y protocolo) para cada servicio, y debemos asegurar que en este archivo se encuentre la definición del servicio TSAP (Punto de Acceso al Servicio de Transporte) ya que esto habilita ISODE a utilizar un puerto tcp especifico (102 en este caso) para el servicio tsap, tal como lo muestra la Figura #11, este archivo se encuentra ubicado dentro de la carpeta “ /etc/ “.

```
...⇒⇒⇒⇒⇒ Línea #45
kerberos88/tcp
supdup  95/tcp
hostnames       101/tcp
iso-tsap 102/tcp        → DEBEMOS ASEGURAR QUE EXISTA ESTA LINEA
x400            103/tcp
x400-snd        104/tcp

csnet-ns        105/tcp

. . .
```

Figura #11 Edición del Archivo services

3.3.1.6.2 Archivo rc.local: Los compiladores, ejecutables, archivos cabecera y ayudas de ISODE ya están listos para poder trabajar con ellos y de esta manera OSIMIS podrá utilizar los servicios OSI que estos le van a prestar, por lo tanto solo nos hace falta colocar en ejecución el demonio tsapd de ISODE, el cual escuchara todas las peticiones entrantes y ejecutara otro demonio adecuado para atender la petición, esto lo hacemos editando el archivo rc.local ubicado en "/etc/rc.d/" y agregando en este las líneas que se muestran en la Figura #12, permitiendo de esta forma que el demonio arranque al iniciar la máquina.

```
...⇒⇒⇒⇒⇒ Línea #6
if [ -f /etc/redhat-release ]; then
        R=$(cat /etc/redhat-release)
else
        R="release 3.0.3"
fi

if [ -f /usr/sbin/tsapd ]; then         → DEBEMOS ADICIONAR
        /usr/sbin/tsapd > /dev/null 2>&1 &  →
        (echo –n 'tsap') > /dev/console     →  ESTAS LINEAS
fi                              →
. . .
```

Figura #12 Edición del Archivo rc.local

3.3.1 ISODE-RPM.

El formato RPM del Red-Hat es un formato que nos presenta los paquetes de manera precompilada y listos para instalar y desinstalar de forma agradable y sencilla, al cumplir entonces con los requerimientos software dados anteriormente en el punto 3.2 la instalación de ISODE en formato RPM no presentara ningún problema y se efectuará en los pasos muy sencillos dados a continuación.

3.3.2.1 Instalación.

La versión del paquete RPM que usamos es "isode8.02.i386.rpm", el cual se encuentra en el CD que acompaña la monografía o se puede obtener en el siguiente enlace en Internet: http://rpmfind.net/linux/RPM/contrib/libc6/i386/isode-8.0-2.i386.html, para realizar la instalación del paquete necesitamos entonces tener permisos de súper-usuario, copiar el archivo al sistema y ejecutar la línea siguiente:

" *rpm –ivh isode8.02.i386.rpm* "

de esta forma quedara instalado adecuadamente la plataforma ISODE, y así podremos observar que archivos se instalaron y en que directorios mediante el comando:

"*rpm –ql isode8.0*".

3.3.2.2 Ajuste y Actualización del sistema.

Ya podemos empezar a ejecutar el demonio de transporte de ISODE pero igual que al instalar ISODE-TAR es aconsejable editar el archivo rc.local que se encuentra en "/etc/rc.d/" agregando las líneas mostradas en la Figura #12, para que así el demonio arranque al iniciar la máquina, notemos que es necesario

saber la ubicación de los archivos que instalo el RPM pues debemos dar la ruta correcta del demonio tsapd.

3.4 CONFIGURACION, COMPILACION E INSTALACION DE OSIMIS.

En los apartados siguientes desglosaremos todos los pasos necesarios para lograr colocar en funcionamiento la plataforma OSIMIS, tal como lo hicimos anteriormente con ISODE.

3.4.1 DESCOMPRESION Y APLICACIÓN DE PARCHES.

La versión de OSIMIS que vamos a instalar es la 4.0, la cual se encuentra en el CD que acompaña la monografía o puede ser obtenida desde la University College of London en la dirección http://www.cs.ucl.ac.uk/research/osimis, la cual se encuentra comprimida en formato TAR y acompañada de varios parches como podemos observar en la Tabla #7.

	Archivos	**Tamaño**
Archivo Principal	osimis-4.0.tar.Z	4568 Kb
Parche # 1	osimis-4.0-patch-01.tar.Z	265 Kb
Parche # 2	osimis-4.0-patch-02.tar.Z	61 Kb
Parche # 3	osimis-4.0-patch-03.tar.Z	158 Kb
Parche # 4	osimis-4.0-patch-04.tar.Z	431 Kb
Parche # 5	osimis-4.0-patch-05.tar.Z	1081 Kb
Parche # 6	osimis-4.0-patch-06.tar.Z	1527 Kb
Parche # 7	osimis-4.0-patch-07.tar.Z	223 Kb
Parche # 8	osimis-4.0-patch-08.tar.Z	155 Kb
Parche # 9	osimis-4.0-patch-09.tar.Z	334 Kb

Tabla #7 Archivos Comprimidos de OSIMIS

El fichero osimis-4.0.tar.Z es el principal, donde están todos los archivos base, el resto son los parches que se han ido liberando a medida que se le hicieron mejoras y arreglaron errores o defectos en los programas (“bugs”), por lo tanto es necesario aplicar todos los parches para el mejor funcionamiento posible de OSIMIS. La forma de descomprimir y aplicar los parches es muy sencilla puesto que solo hay que descomprimir los archivos todos en el mismo directorio, ahorrándonos la necesidad de utilizar la utilidad patch de Linux, entonces los pasos a seguir son:

1. Copiamos los archivos dentro de un directorio, en nuestro caso en /tesis.
2. Ahora los descomprimimos usando: *“ zcat osimis.tar.z | tar xvf –“*
3. Luego aplicamos los parches usando “ *zcat osimis***.tar.z* | *tar xvf* –“ donde *** toma los valores de 1 hasta 9 en orden para aplicar correctamente desde el primero al último parche.

Luego de realizar los pasos anteriores se habrá creado un directorio llamado “/osimis” dentro de “/tesis” donde se encuentran todos los archivos fuente necesarios para compilar OSIMIS, por lo tanto desde ahora expresaremos el estar dentro de este directorio con el símbolo $, es decir: “ /tesis/osimis=$ ”.

3.4.2 ADECUACION DEL AMBIENTE.

Ahora empezamos con el trabajo de configurar y realizar los cambios necesarios para la compilación correcta de la plataforma, para lo cual necesitamos sobrescribir y mover algunos archivos, y editar un archivo especial llamado CONFIG.make, esto es lo que estudiaremos a continuación.

3.4.2.1 Selección de la Versión de ISODE: Primero que todo, necesitamos configurar OSIMIS para la versión de ISODE que estamos usando, OSIMIS nos da la opción de trabajar con las versiones 7.0, 8.0 e IC-R1.0(la cual es la primera liberación del “ISODE Consortium Limited”); por lo tanto debemos seleccionar la versión 8.0 lo cual hacemos trabajando dentro del directorio “$/include”.

1. Primero removemos el directorio isode que trae por defecto OSIMIS usando : **“ *$ rm –rf ./include/isode* “**
2. Luego creamos un enlace con la versión adecuada es decir la 8.0 en este caso, usando: ***“ $ ln –s ./include/isode-8 ./include/isode* “**.

Ahora debemos seleccionar la versión correcta del punto de acceso al servicio de transporte según la versión de ISODE que usemos, esto lo hacemos dentro del directorio “$/misode” así:

1. Removemos el directorio tsap que existe por defecto **“ *$ rm –rf ./misode/tsap* “**
2. Luego creamos un enlace con la versión 8.0 **“ *$ ln –s ./misode/tsap-8 ./misode/tsap* “**

3.4.2.2 Selección del Sistema Operativo: Ahora debemos ir dentro del directorio “$/include/isode”, borrar el archivo config.h. existente allí y realizar un enlace simbólico al mismo archivo con la extensión que indica el sistema operativo en el que vamos a trabajar, en nuestro caso Linux, esto lo hacemos usando: ***“ $ ln – s ./include/isode/config.h.linux ./include/isode/config.h* “**

3.4.2.3 Archivo CONFIG.make: Para efectuar la compilación OSIMIS utiliza unos archivos llamados “Makefile”, los cuales están organizados de tal forma que los cambios requeridos para reflejar la necesidad de un ambiente particular, tal como la ubicación de archivos cabecera y librerías requeridas, nombres de compiladores, directorios de instalación etc, se condensan en un solo archivo, este archivo es llamado CONFIG.make y se encuentra en el directorio “/tesis/osimis”; también cada directorio dentro de “$/osimis” que contiene un Makefile contiene un archivo llamado “make” el cual se encarga de invocar al CONFIG.make antes del Makefile local, de tal forma que el Makefile hereda el ambiente local configurado en CONFIG.make

Como sabemos OSIMIS al igual que ISODE puede ser instalado en diferentes sistemas operativos, por lo tanto nos proporciona un CONFIG.make para cada sistema operativo diferente, estos archivos son:

- CONFIG.make.hpux
- CONFIG.make.linux
- CONFIG.make.solaris
- CONFIG.make.sunos

De estos archivos debemos seleccionar el adecuado para nuestro sistema operativo, esto lo hacemos realizando los siguientes pasos:

1. Borramos el archivo existente por defecto usando: **" *$ rm CONFIG.make* "**.
2. Luego creamos un enlace con el archivo que señala el sistema correcto **" *$ ln –s CONFIG.make.linux CONFIG.make* "**.

A continuación veremos todos los cambios necesarios que debemos realizar para configurar correctamente este archivo.

3.4.2.3.1 Opciones de Compilación Condicionales: Varias de las adiciones que posee la versión 4.0 de OSIMIS están disponibles como opciones de compilación condicionales, y estas pueden ser manipuladas en el archivo CONFIG.make.
Las siguientes son las opciones entre las que podemos seleccionar:

BROWSER:	El navegador de la MIB
LT:	El servicio de ubicación transparente basado en X.500
MONMET:	El monitor métrico de objetos.
MONSUP:	El soporte de monitoreo de objetos.
IQA:	El Internet Q-Adaptor (proxy entre CMIS/P y SNMP)
XMP:	La pila CMIS/P para XOM/XMP
TCLRMIB:	El servicio de soporte de gestión genérico basado en TCL
TPMIB:	El protocolo de transporte MIB para ISODE.
ODPOBJ:	El ejemplo ODP de objetos.

Estas opciones las podemos seleccionar y deseleccionar editando el archivo CONFIG.make usando la marca comentario "#" al comienzo de cada línea (esto nos evita borrar cada línea para no perder información para posibles cambios posteriores). Por ejemplo, para incluir el proxy IQA[24]las siguientes líneas no deben tener la marca # al inicio:

```
#CCONFIG        +=-DIQA
#IQA            = iqa
#IQA_SNTX_LIB   = $(TOP)/proxy/iqa/proxy/libiqasyntaxes.a
```

Cada opción es acompañada por una directiva "CCONFIG +=" que es usada luego por el software para dirigir las opciones de compilación condicionales.
De las opciones de compilación nosotros no elegimos las mostradas en la Tabla #8 (es decir les colocamos la marca # al inicio de cada línea) :

Opción no seleccionada	Razón para no seleccionarla
BROWSER	Debido a que esta diseñado solo para trabajar con SUNOS necesitando Interviews 2.6, y por lo tanto no funciona en Linux.
LT	Puesto que necesita del servicio de directorio (QUIPU) de ISODE para funcionar correctamente, y este no se logro hacer trabajar adecuadamente como ya se hizo notar.
XMP	Solo es necesario para el caso que se quiera usar la pila XOM/XMP de sun versión 8.1 en lugar de ISODE.
TCLRMIB	Es para utilizar la herramienta TCL/TK que permite escribir interpretes de gestores con GUI's, pero las versiones soportadas por OSIMIS de TCL/TK no corren sobre nuestro sistema operativo.

Tabla #8 Opciones no Seleccionadas

El resto de las opciones son seleccionadas para ser compiladas posteriormente, la Figura #12A muestra como debe ser modificado el archivo CONFIG.make y seleccionadas las opciones.

```
...⇒⇒⇒⇒⇒ Línea #48

# LT - X.500-based
#CCONFIG                    += -DUSE_X500
#LT                 = lt
#LT_LIB             = $(TOP)/lt/liblt.a
#X500_LIB           = $(ISODE)/libdsap-osisec.a
#X500_LIB           = $(ISODE)/libdsap.a

# MONMET - generic monitor metric classes as in X.739
#
CCONFIG                     += -DMONMET
MONMET_MIB    = monmet_mib
MONMET_MIB_LIB          = $(TOP)/agent/monmet_mib/libmonmetmib.a
MONMET_SNTX_LIB       = $(TOP)/agent/monmet_mib/libmonmetsntx.a

# MONSUP - generic monitoring support classes combining X.738/X.739
#
CCONFIG        += -DMONSUP
MONSUP_MIB    = monsup_mib
MONSUP_MIB_LIB  = $(TOP)/agent/monsup_mib/libmonsupmib.a

# TCLRMIB - TCL-based generic manager infrastructure for interpreted
# managers
#
#TCLRMIB         = tcl-cmis
#TKCOORD         = TkCoord.o

# IQA - the Internet Q-Adaptor (generic CMIS/P->SNMPv1 proxy)
#
CCONFIG                     += -DIQA
IQA                 = iqa
IQA_SNTX_LIB   = $(TOP)/proxy/iqa/proxy/libiqasyntaxes.a

# XMP - support for Sun XOM/XMP CMIS/P stack version 8.1
#
#XMP                 = msap-xmp

# TPMIB - the Transport Protocol MIB for the ISODE TP0 implementation
#
CCONFIG        += -DTPMIB
TP_MIB = isode_mib
TP_MIB_LIB       = $(TOP)/agent/isode_mib/libisodemib.a
TP_SNTX_LIB     = $(TOP)/agent/isode_mib/libisodesntx.a

# ODPOBJ - the example ODP statistical object
#
CCONFIG        += -DODPOBJ
ODP_OBJ          = odp
ODP_SNTX_LIB = $(TOP)/examples/odp/libodpsntx.a
. . .
```

Figura #12A Edición del Archivo CONFIG.make

3.4.2.3.2 Ajuste del Ambiente Local: Es necesario también realizar ajustes en el archivo CONFIG.make para reflejar la configuración local. Los siguientes caminos (paths) deben ser por lo tanto editados:

HOME: Es el camino absoluto del directorio donde esta osimis $(TOP) elcualcontiene el archivo CONFIG.make.

CC : El compilador de C en nuestro caso (Linux) es el gcc de GNU

CCPLUS: El compilador de C++ el cual es el g++ de GNU

PEPSY : Debe apuntar al compilador ASN.1 pepsy que contiene ISODE.

ISODE : El directorio de las librerías de ISODE.

X11 : El directorio de la librería Xwindows, no es necesario a menos que se desee compilar el navegador (browser).

TCLTK: El directorio de la librería tcl/tk; se necesitará solo si es seleccionada la opción TCLRMIB, por lo tanto no se necesitara en nuestro caso.

TCLINC: El directorio del sistema local que contiene el archivo cabecera TCL. No es necesario en nuestro caso, por lo tanto puede tener cualquier valor, o ser puesto en comentarios.

TKINC: El directorio del sistema local que contiene los archivos cabecera TK. No es necesario en nuestro caso, por lo tanto puede tener cualquier valor, o ser puesto en comentarios.

La Figura #12B presenta las modificaciones necesarias que debemos realizar en el archivo para incluir los anteriores ajustes.

```
...⇒⇒⇒⇒⇒ Línea #105

# The following part configures the local environment.
#

HOME    =       /tesis/osimis

# LSOCKET should be as in the ISODE CONFIG.make
#
LSOCKET =

# GNU C version 2.7.2
CC      =       /usr/bin/gcc
```

```
# GNU C++ versión 2.7.2
CCPLUS =        /usr/bin/g++

PEPSY   =       /usr/bin/pepsy

ISODE   =       /usr/lib

X11     =       -L/usr/X11R6/lib
TCLTK   =       /usr/lib

# the next is only needed for building the optional kernel/TkCoord.o
#
TCLINC =        /usr/include
TKINC   =       /usr/include
. . .
```

Figura #12B Edición del Archivo CONFIG.make

3.4.2.3.3 Directivas de Compilación: Ahora miremos ciertas directivas importantes para los compiladores que deben ser editadas dentro del mismo archivo CONFIG.make, estas directivas son:

CCONFIG += : Como dijimos anteriormente esta directiva es usada para dirigir las opciones de compilación, como no necesitamos usar la pila CMOT[25] debemos asegurarnos que en la lista de CCONFIG la directiva –DLPP no este presente o se encuentre en comentarios; También es bueno aclarar que podemos usar OSIMIS en un PC con Linux stand–alone (sin red), para lo cual debemos adicionar –DLOCALHOST a la lista CCONFIG para que sea capaz de trabajar sobre la interfaz local (loopback), pero para nuestro caso en el cual trabajamos en la LAN de la Universidad del Cauca no debe estar seleccionada esta opción.

LDFLAGS : Nos permite obtener los programas mas limpios y de tamaño pequeño, para lo cual debemos asegurarnos que este fijado con la opción “-s”.

LPP: Es una directriz para usar la capa de presentación ligera directamente sobre TCP (Pila CMOT), en tal caso se debe fijar con la opción “-lpp”, pero nosotros no la usaremos así que debe ser colocada en comentarios.

CMPL : Esta opción se puede usar si se desea construir OSIMIS para más de un compilador, por ejemplo GNU y ATT C++, pero en nuestro caso debe ser vacía.

ARCH : En forma parecida a la anterior podemos usar ésta si deseamos construir OSIMIS para más de una arquitectura de máquina por ejemplo SUN4 y SUN3, SUN4 y RS6000 etc, pero en nuestro caso debe ser vacía.

Los cambios que debemos realizar en el archivo para modificar adecuadamente las directivas anteriores son mostrados en las Figuras #12C y #12D

```
...⇒⇒⇒⇒⇒ Línea #163

CCONFIG +=      -DDEBUG -DLOGGING -DPERSISTENCY #-DLOCALHOST #-DTIMING #-DLPP
INC       =
OPTIONS =       -I. -I$(OSIMIS-INC) \
                -I$(OSIMIS-INC)/isode/pepsy \
                -I$(OSIMIS-INC)/isode/quipu \
                -I$(H) -I$(TCLINC) $(CCONFIG)
CCFLAGS         =       $(OPTIONS) -g # -O
LDFLAGS         =       -s
. . .
```

Figura #12C Edición del Archivo CONFIG.make

```
...⇒⇒⇒⇒⇒ Línea #131
TCLINC=         /usr/include
TKINC   =       /usr/include

#LPP    =       -lpp
CMPL    =
#CMPL   =       .$(CCPLUS)
ARCH    =
#ARCH   =       .sun3
#ARCH   =       .sun4
. . .
```

Figura #12D Edición del Archivo CONFIG.make

3.4.3 MODIFICACIONES AL CODIGO FUENTE.

Antes de entrar a discutir sobre la compilación de la plataforma debemos efectuar algunos cambios en el código fuente, estos cambios serán presentados a continuación.

3.4.3.2 Archivo isodeICC.cc.

3.4.3.2.1 Descripción: El archivo isodeICC.cc es una fuente de conocimiento especifica (lo cual será aclarado en el Capitulo IV) que se encarga de la comunicación entre un agente de OSIMIS y la capa de transporte que implementa ISODE.

3.4.3.2.2 Ubicación: El archivo se encuentra en la ruta "/tesis/osimis/agent/isode_mib/".

3.4.3.2.3 Modificaciones: Las modificaciones necesarias a realizar en este archivo son mostradas en la Figura #13.

```
...⇒⇒⇒⇒⇒ Línea #110
int  IsodeICC::readCommEndpoint (int port)
{
   MReport minfo;
   transportConnection* tpConn;
   char id[32];
   RDN rdn;

   read(port, (char*) &minfo, sizeof(MReport)); → BORRAR ESTA LINEA

   recvfrom(port, (char*) &minfo,sizeof(MReport),0,NULL,NULL); → ADICIONAR ESTA LINEA


   switch(minfo.type) {
   case OPREQIN:
   case OPREQOUT:
…
```

Figura #13 Edición del Archivo isodeICC.cc

3.4.3.2.4 Justificación de las Modificaciones: La Figura #13 nos muestra que las modificaciones dentro del archivo isodeICC.cc las realizamos en la función readCommEndpoint, la cual se encarga de leer un socket en el cual escribe ISODE en el momento en que algún evento importante ocurre en la capa de transporte(como realizar una conexión, desconexión etc), pero vemos que en el código original se utiliza la función "read()" de Linux para realizar tal lectura, lo cual es un error que logramos descubrir al estudiar el código fuente de ISODE, pues este utiliza la función "sendto()" de Linux usada para enviar datos sobre sockets no

conectados de datagramas, los cuales no pueden ser leídos usando “read()” si no que se debe usar la función “recvfrom()”, esta es la razón del cambio en el código del archivo isodeICC.cc.

3.4.4 COMPILACIÓN.

Antes de poder comenzar con el procedimiento de compilación, necesitamos fijar una variable de ambiente que apunte al directorio /etc de OSIMIS puesto que este no se enlaza rígidamente en el código, esta variable debe tener el nombre de OSIMISETCPATH. Este paso se hace de la siguiente forma usando el shell bash de UNIX: **“ *$ export OSIMISETCPATH=/tesis/osimis/etc* “.**

En este punto ya podemos realizar la compilación, lo cual hacemos con el siguiente comando:

“ *$ ./make all* “ o si deseamos enviar las acciones a un archivo para su posterior estudio usamos

“ *$ ./make all > ♦♦♦ 2>&1* ” donde ♦♦♦ es el nombre del archivo.

Esta última opción es más recomendable ya que nos permite observar con detenimiento las advertencias (“warnings”) y posibles errores que sucedan por olvidársenos o saltarnos algún paso o por el contrario para observar la correcta compilación puesto que son bastante extensos los resultados desplegados.

En caso de alguna equivocación y que por tanto no sea correctamente compilado OSIMIS, debemos usar el siguiente comando para limpiar los archivos compilados e incorrectos: **“ *$ ./make clean* “** antes de intentar compilar de nuevo la plataforma.

3.4.4.1 Compilando misode: A fin de poder gestionar la implementación de la capa de transporte, algunas herramientas de gestión deben ser adicionadas por OSIMIS a la capa de transporte implementada por ISODE, así el directorio

“/tesis/osimis/misode/“ contiene una versión de la librería ISODE con las herramientas compiladas allí, este directorio también contiene una versión del test cliente/servidor estándar de ISODE y los programas imisc/imiscd enlazados mediante aquella librería como aplicaciones de gestión simple, por lo tanto se renombran a mimisc y mimiscd, estas aplicaciones funcionan correctamente y pueden ser usadas sin ningún problema.

La aplicación de gestión de la capa de transporte que posee OSIMIS, desafortunadamente no viene lista para compilar e instalar usando el archivo CONFIG.make, por lo tanto debe ser compilada aparte, de la siguiente forma:

nos ubicamos dentro del directorio “/tesis/osimis/misode/” y ejecutamos

“ *./make all* ” o si deseamos enviar las acciones a un archivo para su posterior estudio usamos

“ *./make all > ♦♦♦ 2>&1* ” donde ♦♦♦ es el archivo en el que deseamos guardar las acciones de instalación realizadas.

El uso de cualquiera de los anteriores comandos nos generara ciertos “warnings” que son inocuos, y los binarios necesarios serán creados.

3.4.5 INSTALACIÓN.

Cuando OSIMIS es compilado, las librerías y programas permanecen en sus directorios dentro del árbol fuente, en el procedimiento de instalación se mueven estos a los directorios deseados que se especifican en el archivo /tesis/osimis/CONFIG.make. Nosotros podemos seleccionar estos directorios en sus lugares por defecto, los cuales están dentro del árbol fuente, pero también podemos escoger directorios ubicados en cualquier otro sitio; como este es

nuestro caso ya que es deseable dejar los ejecutables aparte del código fuente, entonces debemos cambiar los caminos que traen por defecto los siguientes directorios:

LIB : Éste es el directorio donde las librerías de OSIMIS deben ser instaladas.

BIN: Este es el directorio donde los programas de OSIMIS deben ser instalados.

OSIMIS – INC: Este es el directorio donde las adiciones y reemplazos de OSIMIS para los archivos cabecera de sistemas son instalados.

H : Este es el directorio donde los archivos cabecera de OSIMIS deben ser instalados.

ETC: Este el directorio ETC de OSIMIS.

MAN: Este es el directorio donde las páginas del manual de OSIMIS se instalan.

La Figura #14 especifica como modificar el archivo CONFIG.make para seleccionar los directorios que nosotros utilizamos.

```
...⇒⇒⇒⇒⇒ Línea #155

LIB       =          /osimis/lib
BIN       =          /osimis/bin
OSIMIS-INC =  /osimis/include
H         =          /osimis/include
ETC       =          /osimis/etc
MAN       =          /usr/man


CCONFIG +=      -DDEBUG -DLOGGING -DPERSISTENCY #-DLOCALHOST #-DTIMING #-DLPP
. . .
```

Figura #14 Edición del Archivo CONFIG.make para Instalación.

Ahora si realizamos la instalación de OSIMIS siguiendo los pasos que a continuación se muestran.

1. " $ ./make install-etc >♦♦♦2>&1 "

2. “ $ ./make install-h >♦♦♦2>&1 ”
3. “ $ ./make install-prog >♦♦♦2>&1 ”
4. “ $ ./make install-inc >♦♦♦2>&1 ”
5. “ $ ./make install-lib >♦♦♦2>&1 ”
6. “ $ ./make install-man >♦♦♦ 2>&1 ”
7. “ $ ./make install-bin >♦♦♦2>&1”

donde ♦♦♦ es el nombre del archivo en el que deseamos guardar las acciones realizadas para su posterior estudio.

3.4.5.1 Instalando misode: Como hicimos notar anteriormente es necesario instalar la aplicación de gestión del protocolo de transporte aparte de la de OSIMIS, por lo tanto el proceso de instalación de esta puede hacerse en los siguientes pasos:

1. “ $ cd misode ”
2. “ ./make install-lib > ♦♦♦ 2>&1 ”
3. “ ./make install-prog >♦♦♦2>&1 “
4. “ ./make install-man > ♦♦♦2>&1 “
5. “ ./make install-bin > ♦♦♦2>&1 “

donde ♦♦♦ es el nombre del archivo en el que deseamos guardar las acciones realizadas.

3.4.6 AJUSTE Y ACTUALIZACIÓN DE LAS BASES DE DATOS.

Luego de haber conseguido la compilación e instalación exitosa de OSIMIS, necesitamos ajustar las bases de datos OSIMIS/ISODE para reflejar la configuración local. ISODE usa una base de datos que mantiene información de varias cosas por ejemplo pilas de servicios de transporte, direcciones de aplicación, tablas con identificación de objetos y posible sintaxis ASN.1 asociada

etc. Este directorio es conocido como ETCDIR el cual es enlazado rígidamente en la librería cuando éste es construido.
OSIMIS necesita usar un directorio con exactamente la misma funcionalidad como el ETCDIR de ISODE pero los dos directorios (de ISODE y OSIMIS) no pueden ser combinados debido a la colisión de nombres que se generaría, así OSIMIS usa uno exclusivo. En OSIMIS, debemos usar uno que no este enlazado rígidamente en la librería cuando esta es construida, más bien usamos una variable apuntando hacia él a fin de incrementar la flexibilidad, esta es la razón para usar el OSIMISETCPATH que habíamos nombrado antes.
Ahora miremos las diferentes bases de datos que debemos actualizar.

3.4.6.1 Ajustando el archivo isoentities: Primero que todo debemos actualizar aquellas partes de la base de datos asociadas al direccionamiento de aplicaciones las cuales son usadas por defecto cuando el servicio de directorio (X.500) no es usado, como es nuestro caso. Por lo tanto debemos editar el archivo isoentities ubicado en "/osimis/etc/" para incluir las direcciones de presentación de los agentes de gestión dentro de los "hosts" en los cuales ellos se ejecutarán. Necesitamos adicionar entradas para cada "host" en la red en el cual deseemos correr los agentes de OSIMIS, a continuación mostramos en que consiste una entrada en el archivo isoentities:

```
osimis          SMA         1.17.5.12.0  \
                #603/internet=200.21.83.142+11010
```

donde:

osimis	Es el nombre del "host".
SMA	La aplicación de gestión (nombre lógico)
1.17.5.12.0	El identificador de objeto calificador de la aplicación.
#603	El selector de transporte.
Internet	La dirección de red de la comunidad.
200.21.83.142	dirección IP del sistema 'osimis'

11010 El puerto TCP de aplicación SMA.

Debemos asegurarnos que el puerto TCP usado sea un número local único en cada sistema final que se vaya a usar a fin de evitar colisiones. Para tal efecto debemos chequear el archivo “services” ubicado en “/etc” para observar posibles puertos TCP que tengan el mismo número, y si este es el caso, se modifica el de OSIMIS por uno no usado. La Figura #15 aclara lo anterior y muestra las entradas usadas por nosotros en el archivo isoentities.

```
...⇒⇒⇒⇒⇒ Línea #1

##############################################################################
# ISOENTITIES DATABASE FOR OSIMIS
##############################################################################

osimis    SMA          1.17.5.12.0  \
                #603/Internet=200.21.83.142+11010

tao       SMA          1.17.5.12.0  \
                #603/Internet=200.21.83.185+11010


osimis    IQA          1.17.5.14.0  \
                #605/Internet=200.21.83.142+11012

tao       IQA          1.17.5.14.0  \
                #605/Internet=200.21.83.185+11012

osimis    ODPTEST      1.17.5.15.0  \
                #606/Internet=200.21.83.142+11013

tao       ODPTEST      1.17.5.15.0  \
                #606/Internet=200.21.83.185+11013

osimis    AGENTE           1.17.5.16.0  \
                #607/Internet=200.21.83.142+11014

tao           AGENTE       1.17.5.16.0  \
                #607/Internet=200.21.83.185+11014

# end of isoentities database
```

Figura #15 Edición del Archivo isoentities.

3.4.6.2 Ajustando el archivo osimistailor: Existe un archivo especial en el ETCDIR de OSIMIS llamado osimistailor ubicado en “/osimis/etc/”. En este archivo necesitamos fijar las variables relacionadas con el “logging”, estas son gmsLogDir y gmsLogRecordAccess. El gmsLogDir especifica donde los agentes deben almacenar los “logs” u otros objetos de gestión persistentes, el directorio que seleccionemos debe existir y debemos asegurarnos de no seleccionar un directorio temporal del sistema pues perderíamos la información. La siguiente variable que debemos fijar es la bandera gmsLogRecordAccess la cual puede tener los valores “on” u “off”, esta variable permite la operación de scoping para que el registro log sea conmutado “on” u “off” siendo este el mecanismo de control de acceso usado para prohibir el acceso al registro log a través del “scoping” de CMIS. Los objetos alcanzados (scoped) cuando se realiza gestión deben estar disponibles lo cual significa posible agotamiento de memoria para “logs” con miles de registros (dependiendo de la memoria disponible y el espacio “swap”). Si se decide usar OSIMIS para monitorear recursos que generen muchísimas notificaciones que puedan ser “logged", es recomendable conmutar esta variable a “off”.

La Figura # 16 muestra los valores que le debemos dar a estas variables.

```
...⇒⇒⇒⇒⇒ Línea #24
# the following path should not be /tmp
gmsLogDir:        /osimis/logs
gmsLogRecordAccess: on
gmsGlobalNameDomain: c=GB@o=UCL@ou=CS
gmsGlobalNames:              off
```

Figura #16 Edicion del Archivo osimistailor.

3.4.6.3 Ajustando el archivo isobjects: El archivo isobjects ubicado en “/osimis/etc” es el que nos permite realizar el mapeo entre descriptores de objetos e identificadores de objetos, dándonos la compatibilidad con los objetos de ISODE

puesto que define sus identificadores de objeto para las aplicaciones del protocolo de transporte, FTAM, X.500, etc.

En este archivo debemos definir el identificador de objeto para cada agente existente y por lo tanto debemos adicionar una línea para cada nuevo agente que vayamos a incorporar, la Figura #17 nos muestra como adicionar un nuevo agente en este archivo.

```
...⇒⇒⇒⇒⇒ Línea #9
"SMA"                   1.17.5.12.0
"IQA"                   1.17.5.14.0
"ODPTEST"               1.17.5.15.0
"finalagente"    1.17.5.16.0   → ADICIONAR ESTA LINEA

################################################################################
#
# isobjects - ISODE Objects Database
. . .
```

Figura #17 Edición del Archivo isobjects.

3.4.6.4 Ajustando el archivo iqastartupagents.icm: Para que el proxy de gestión IQA funcione correctamente debemos realizar ciertos pasos; primero debemos asegurarnos que el directorio iqadir existe en el directorio ETCDIR de OSIMIS el cual debe haber sido copiado en la instalación ya que este contiene archivos necesarios para el arranque del agente, luego debemos editar el archivo iqastartupagents.icm ubicado en el directorio "/osimis/etc/iqadir**/**" eliminando cualquier información que este archivo tenga por defecto, y adicionando la información mostrada en la Figura #18.

```
...⇒⇒⇒⇒⇒ Línea #1
# IQASTARTUPAGENTS.ICM

"osimis",{snmpUDPDomain:0xc8:15:53:8e:00:a1}, snmpV1, {IIMCrfc1213}, / 3, 2000, "public", 0, 3163, "traps"

"tao",     {snmpUDPDomain:0xc8:15:53:b9:00:a1}, snmpV1, {IIMCrfc1213}, / 3, 2000, "public", 0, 3163, "traps"

#end
```

Figura #18 Edición del Archivo iqastartupagents.icm

Haciendo referencia a la información de la Figura #18, los primeros ocho parámetros de configuración son obligatorios para habilitar el proxy con un agente SNMP remoto y significan lo siguiente:

1. El sistema donde se encuentra el agente remoto.
2. La pareja “dominio de transporte/dirección de transporte” donde la dirección IP de la máquina y el puerto a usar están y deben ser expresados en hexadecimal, como vemos para la máquina osimis con IP 200.21.83.142 y el puerto 161 el valor hexadecimal es 0xc8:15:53:8e:00:a1 (usando 4 octetos para la dirección IP y 2 para el puerto).
3. El protocolo de gestión remoto, en esta versión de OSIMIS sólo es soportado el SNMPv1.
4. El OID de la MIB que es soportada remotamente, es posible soportar múltiples MIBs y deben ir aquí separadas por un espacio.
5. El número máximo de retransmisiones sobre el intervalo de una petición remota.
6. El intervalo de retransmisión en milisegundos.
7. La cadena de la comunidad requerida por el agente remoto.
8. Un número entre 0 y 2.

   Fijado a 0: No intenta recuperar comunicaciones fallidas con agentes.

   Fijado a 1: este permite que el IQA intente recuperar las comunicaciones con agentes que no retornan “TooBig” a las peticiones SNMP.

   Fijado a 2: entonces el IQA puede mantener una ruta a los objetos que un agente remoto no soporta y no intentara recuperar este de nuevo.

NOTA: El carácter “/” es usado para partir las líneas demasiado largas como en la Figura #18.

El par <puerto>, <comunidad del trap> (3163, ”traps”) es opcional para soportar los traps, aquellos que son aceptados por el IQA están definidos en el archivo “ /tesis/osimis/proxy/iqa/pma/pma.h”

3.4.6.5 Ajustando los archivos oidtable: Existen tres archivos oidtable ubicados dentro del directorio “/osimis/etc” los cuales debemos modificar, estos archivos son mostrados en la Tabla #9:

oidtable.at:	Contiene las tablas OID de gestión OSI para los atributos, acciones y notificaciones.
oidtable.gen:	Contiene las tablas OID de gestión OSI para etiquetas genéricas, clases de objetos, nombres enlazados y atributos de paquetes y grupo.
oidtable.oc:	Contiene las tablas OID del directorio OSI para las clases de objetos.

Tabla #9 Archivos oidtable.

La modificación de estas bases de datos se debe a que el proxy IQA debe adicionar sus propios atributos, acciones, notificaciones, clases de objeto etc, las cuales han sido implementadas por el trabajo del IIMC (Internet Management Coexistence), entonces lo que debemos hacer, puesto que el IQA se compila automáticamente al compilar OSIMIS, es reemplazar los oidtable que se encuentran el ETCDIR de OSIMIS por los generados por el IQA, es decir:

1. Borramos los oidtable del ETCDIR asi:**“ *rm /osimis/etc/oidtable** ”**
2. Lluego copiamos los oidtable desde el IQA asi: **“ *cp /tesis/osimis/proxy/iqa/etc/oidtable* /osimis/etc* “**

3.4.6.6 Actualizando el archivo services: Para que el agente SMA que trae OSIMIS incluido pueda realizar la gestión del protocolo de transporte OSI es necesario actualizar el archivo services del “host” donde se correrá el agente de gestión, este archivo esta ubicado en la ruta “/etc”.
La Figura #19 muestra la actualización de este archivo.

```
...⇒⇒⇒⇒⇒ Línea #11


tcpmux  1/tcp       # rfc-1078

manager    6/udp   # SMA          → ADICIONAR ESTA LINEA

echo               7/tcp
echo               7/udp
. . .
```

Figura #19 Edición del Archivo services para usar SMA.

Debemos asegurar que el número del puerto UDP es único en el sistema local, este puerto es necesario porque la pila ISODE es un código enlazado con aplicaciones OSI y corre como parte del proceso de aplicación, así la invocación del protocolo de transporte usa datagramas UDP para reportar información de gestión asincrónicamente al SMA, el cual escucha el puerto anterior (es decir los reportes asincrónicos desde recursos reales a objetos gestionados).

3.4.6.7 Configurando el inicio de OSIMIS: Como ya mencionamos el camino ETCDIR de OSIMIS no está enlazado rígidamente en el código, y la variable de ambiente OSIMISETCPATH debe ser usada en lugar de eso y debe apuntar al directorio ETC de OSIMIS. Aquí lo que haremos es colocar esta en un archivo shell de arranque del sistema Linux para que no se pierda en el caso de resetear la máquina. Esto lo hacemos en el archivo .bash_profile de Linux agregando la línea: “export OSIMISETCPATH=/osimis/etc”

3.5 PROBANDO EL SISTEMA.

En este punto ya podemos acceder a las ayudas en línea, y empezar a utilizar OSIMIS iniciando los agentes SMA, ODPTEST e IQA y podemos entonces usar los programas para conectar a estos agentes, estos programas son mostrados en la Tabla #10, y los agentes son mostrados en la Tabla #11, todos estos archivos ejecutables están ubicados en la ruta “/osimis/bin/”.

Programa Ejecutable	Descripción
mibdump	Habilita la conexión a un agente de gestión OSI remoto y recupera información de gestión, este establece primero una asociación de gestión, pide el objeto gestionado según sea especificado a través de los argumentos de la línea de comandos (usando una petición M-GET de CMIS), imprime la salida y/o error y entonces libera la asociación y sale.
mset	Permite conectarse a un agente de gestión OSI remoto y fijar información de gestión (atributos de objetos gestionados), este establece primero una asociación de gestión, pide que la operación "set" sea desempeñada según sea especificada a través de los argumentos de la línea de comandos (usando una petición M-SET de CMIS), imprime los resultados y/o errores y entonces libera la asociación y sale.
maction	Habilita la conexión a un agente de gestión OSI remoto y desempeña una acción sobre objetos gestionados, este establece primero una asociación de gestión, pide la operación de la acción a ser desempeñada según sea especificada a través de los argumentos de la línea de comandos (usando una petición M-ACTION de CMIS), imprime los resultados y/o errores y entonces libera la asociación y sale.
mcreate	Habilita la conexión a un agente de gestión OSI remoto y crea un objeto gestionado en esta MIB, establece primero una asociación de gestión, pide que la operación de creación sea desempeñada según sea especificada a través de los argumentos de la línea de comandos (usando una petición M-CREATE de CMIS), imprime el resultado y/o error y entonces libera la asociación y sale.
mdelete	Habilita la conexión a un agente de gestión OSI remoto y borra objetos gestionados en su MIB, establece primero una asociación de gestión, pide la operación de acción a ser desempeñada según sea especificada a través de los argumentos de la línea de comandos (usando una petición M-DELETE de CMIS), imprime el resultado y/o error y entonces libera la asociación y sale.
evsink	Habilita la conexión a un agente de gestión OSI remoto y pide y recibe reporte de eventos, este establece primero una asociación de gestión, luego pide el reporte de eventos según sea especificado a través de los argumentos de la línea de comandos, creando un control de gestión del discriminador de envío de eventos (usando una petición M-CREATE de CMIS) y entonces escucha por reportes de eventos los cuales imprime a la salida estándar, el programa termina al recibir una señal SIGQUIT en primer plano (Control+c para la mayoría de key-boards) o una señal SIGTERM en segundo plano (producida por el programa kill de UNIX). Luego de la recepción de la señal de terminación, esta borra el discriminador de eventos y crea y libera la asociación de gestión antes de salir.
evlog	Permite el uso de actividades logging sobre un agente OSI, este permite la creación y borrado de objetos eventLog y objetos eventRecord, y permite fijar

	ciertos valores de atributos del objeto eventLog, controlando su comportamiento.

Tabla #10 Programas Ejecutables de Gestión.

Programa ejecutable del Agente	Descripción
sma	El servidor *sma* es un agente de gestión de sistemas que implementa una MIB (no estándar) para el protocolo de transporte ISO, el cual es usado para gestionar la implementación ISODE del último, este también implementa un ejemplo de la MIB UNIX no estándar con unos objetos de gestión triviales mostrando el número de usuarios en el sistema.
odptestsrv	Este es un agente de gestión proporcionado con la intención de mostrar la posibilidad de usar el modelo de gestión OSI para el procesamiento distribuido, y permite realizar dos acciones muy sencillas, calcular la raíz cuadrada de un número real no negativo y calcular la media y desviación estándar de una serie de números reales.
iqa	Este es el agente de gestión que trabaja como proxy entre SNMP y CMIP y que nos permite realizar gestión sobre MIBs SNMPv1.

Tabla #11 Agentes Ejecutables

Todos los agentes serán extensamente explicados en los Capítulos IV y V, aquí hemos dado una muy breve descripción, y los programas para realizar la gestión pueden ser estudiados en las ayudas en línea que proporciona OSIMIS, y en el Capitulo V daremos algunos ejemplos de cómo usarlos.

CAPITULO IV

COMPONENTES DE OSIMIS

En este capitulo entraremos a estudiar los servicios y aplicaciones (tratados en el Capitulo II) que conforman OSIMIS, los cuales nos ayudaran a comprender como OSIMIS logra implementar los estándares de gestión OSI y TMN, brindándonos toda la potencialidad de estos. Con el estudio de los servicios podremos además conocer la base estructural de la plataforma, lo cual es muy importante para develar como es el funcionamiento interno de OSIMIS.

4.1 LA IMPLEMENTACIÓN CMIP/S.

OSIMIS utiliza el modelo de gestión OSI como el método para realizar gestión "*end-to-end*", por lo tanto implementa el protocolo/servicio de información de gestión común (CMIP/S). Este está implementado como una librería construida en lenguaje C y usa los elementos de servicio ROSE[26] y ACSE[27] que facilita ISODE junto con el soporte ASN.1. La implementación de CMIS/P opera de la siguiente forma:

- Cada petición y respuesta de la primitiva CMIS es realizada a través de una llamada a procedimiento, y las indicaciones y confirmaciones son realizadas a través de una llamada al procedimiento "*wait*". Las asociaciones que se establezcan son representadas como comunicaciones finales (archivos descriptores de UNIX, usualmente sockets) y llamadas al sistema operativo,

por ejemplo la llamada a "s*elect()"* de UNIX es usada para multiplexar éstas, y poder realizar el manejo de eventos.

La API CMIS de OSIMIS es conocida como punto de acceso al servicio de gestión MSAP(Management Service Access Point), la cual se encuentra en "/tesis/osimis/msap" y contiene la librería que implementa la funcionalidad de CMIS/P como se describe en los estándares ISO/IS 9595 versión 2 "Common Management Information Service Specification" e ISO/IS 9596 versión 2 "Common Management Information Protocol Specification", ambos estándares de 1991; las funciones que esta librería proporciona son mostradas en la Tabla # 12.

FUNCION	DESCRIPCION
initialiseSyntaxes	Establece la información de sintaxis de ISODE.
M_AbortReq	Solicita la liberación abrupta de una asociación de gestión.
M_Action/M_ActionConf	Desempeñan una acción de gestión.
M_ActionRes	El resultado de la acción de gestión.
m_initialise, m_terminate	Establece y libera una asociación de gestión.
M_CancelGet	Cancela una petición M_Get previa.
M_CancelGetRes	El resultado de cancelar una petición.
M_Create	Crea un objeto gestionado.
M_CreateRes	El resultado de crear el objeto gestionado.
M_Delete	Borra un objeto(s) gestionado(s).
M_DeleteRes	El resultado de borrar un objeto gestionado.
M_EventRep/M_EventRepConf	Envía un reporte de eventos.
M_EventRepRes	El resultado del reporte de eventos.
M_Get	Solicita información de gestión.
M_GetRes	El resultado de solicitar información de gestión.
M_Init	Inicializa una respuesta de gestión
M_InitialiseReq	Establece una asociación de gestión.
M_InitialiseResp	La respuesta a una petición de asociación de gestión.
M_Set/M_SetConf	Fija información de gestión.
M_SetRes	El resultado de fijar información de gestión.
M_TerminateReq	Envía una petición de liberación de asociación de gestión.

M_TerminateResp	Responde a una petición de liberación de asociación de gestión.

Tabla #12 Funciones Proporcionadas por MSAP

Si se desea un estudio detallado de estas funciones se puede acudir a las ayudas en línea (páginas del manual en línea usando el comando “man”) que proporciona OSIMIS.

MSAP fue concebida mucho antes que el estándar X/Open XMP fuera especificado y por tanto no es fiel a este, siendo diseñada específicamente para CMIS y no para CMIS y SNMP como el XMP, la diferencia, muy ventajosa por cierto, es que ésta oculta más información y puede realizar implementaciones más eficientes; la razón de no haberse implementado completamente en un modelo orientado a objetos (usando C++), es para facilitar la integración con ISODE, el cual está construido en lenguaje C. OSIMIS también incluye una implementación del Internet SNMPv1, la cual es usada por la pasarela de aplicación genérica entre CMIS/P y SNMP. Esta pasarela usa la implementación del UDP que tiene UNIX y el soporte ASN.1 de ISODE, y es implementada en la misma forma que CMIS/P. Las aplicaciones usando CMIS necesitan manipular tipos ASN.1 para los valores de atributos del objeto, las acciones, las notificaciones y los parámetros de error de CMIS.

4.2 EL MECANISMO DE COORDINACIÓN.

La gestión y en general las aplicaciones distribuidas tienen necesidades complejas en términos del manejo de entradas externas, las aplicaciones de gestión tienen adicionalmente necesidad de manejar mecanismos de alarmas internas para ordenar tareas periódicas en tiempo real (“*polling*”, “scheduling”, etc.). Además, algunas aplicaciones pueden necesitar el ser integradas con interfaces de usuario graficas (GUI), las cuales tienen sus propios mecanismos para el manejo de datos

desde el teclado y ratón.

Hay un número diferente de técnicas para permitir a una aplicación manejar eventos externos e internos, y estos eventos necesitan ser manejados de tal forma que los recursos sean usados cuando el sistema está desocupado, las dos técnicas más comunes son:

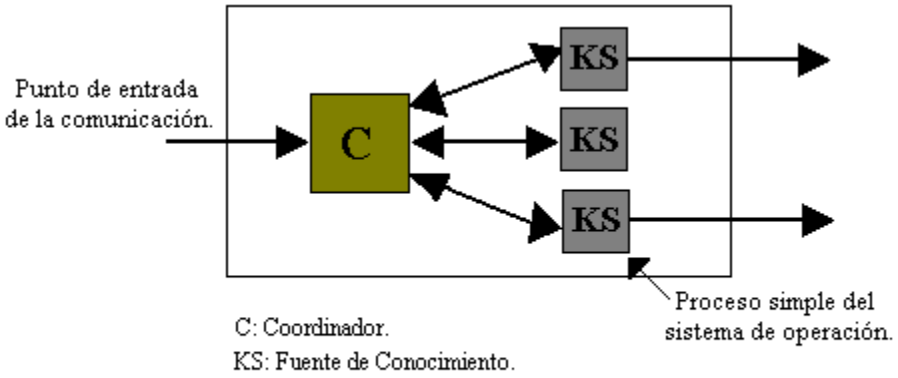
- Usar un hilo de ejecución simple.
- Usar un hilo de ejecución múltiple.

En el primero, las comunicaciones externas deben seguir un modelo asincrónico, puesto que esperan por el resultado de una operación remota, en un modelo sincrónico este bloquearía el sistema completo. Por supuesto, un mecanismo común es necesario para escuchar todas las comunicaciones externas y demultiplexar los datos entrantes, este requerimiento es suministrado por el mecanismo de coordinación de OSIMIS. En el segundo, muchas filas de control pueden ejecutarse simultáneamente (en un modelo seudo-paralelo) dentro del mismo proceso, lo cual significa que el bloqueo sobre un resultado externo es permitido. Este es el estilo de organización usado por las plataformas de sistemas distribuidos puesto que están basadas en RPC[28](Remote Procedure Call), el cual es inherentemente sincrónico con respecto a los objetos cliente que desempeñan operaciones remotas.

La ventaja del primer mecanismo es que este es soportado por muchos sistemas operativos y como tal es liviano y eficiente mientras su desventaja radica en que introduce condiciones en el manejo de los resultados de operaciones remotas asincrónicas. El segundo mecanismo permite una programación más natural sin introducir condiciones con respecto a las operaciones remotas pero este requiere mecanismos de aseguramiento interno y reentrada de código y además no llega a ser muy eficiente. Un problema adicional en organizar una aplicación compleja concierne al manejo de temporizadores de alarmas internas, muchos sistemas de

operación no apilan estos, por ejemplo algunos pueden sólo tener una alarma pendiente por cada proceso, esto significa que se necesita un mecanismo común para asegurar el correcto uso de los mecanismos subyacentes del sistema operativo.

OSIMIS al proveer una infraestructura orientada a objetos en C++ permite a una aplicación ser organizada en un completo modelo manejado por eventos bajo el paradigma de hilo de ejecución simple, donde cada evento externo o interno es colocado en el hilo de ejecución y tratado en la base “primero en entrar – primero en ser atendido”. Este mecanismo permite una fácil integración de entradas de recursos externos adicionales o temporizadores de alarmas y esto es realizado por dos clases C++: la clase Coordinator (coordinador) y la clase KS (Knowledge Source – Fuente de conocimiento); Siempre debe existir solo una instancia de Coordinator o cualquier clase derivada en la aplicación, mientras KS es una clase abstracta que facilita el uso de los servicios de la clase Coordinator e integra entradas de recursos externos y temporizadores de alarmas. Todos los eventos externos y temporizadores de alarmas son controlados por Coordinator cuya presencia es transparente para el implementador de la clase KS especifica ya que es accedida a través de la interfaz KS genérica; este modelo es descrito en la Figura # 20.



Figura#20 El Modelo del Proceso de Coordinación de OSIMIS

Este mecanismo de coordinación (el encargado de manejar todas las entradas externas e internas como se explico anteriormente) está diseñado de tal forma que permite la integración con mecanismos de coordinación de otros sistemas. Esto se logra a través de clases especiales derivadas de Coordinator las cuales trabajaran con el mecanismo particular, estas clases derivadas desde Coordinator aún mantienen el control de los recursos de entrada y temporizadores de alarmas de los KSs específicos de OSIMIS, pero pasan la tarea de desempeñar la escucha central a los mecanismos de coordinación de los otros sistemas.

El hecho de pasar la escucha central a mecanismos de otros sistemas se hace necesario por ejemplo para agentes de OSIMIS que desean recibir peticiones de asociación desde ISODE, ya que éste impone su propio mecanismo de escucha que oculta el punto de acceso al servicio de presentación (PSAP [29]) sobre el cual nuevas asociaciones ACSE son aceptadas, por lo tanto con el fin de recibir asociaciones desde ISODE, se debe derivar una clase desde Coordinator y crear una clase KS especifica, esta clase derivada desde Coordinator a sido incluida dentro del paquete OSIMIS como un ejemplo y existe con el nombre IsodeCoordinator, y la clase KS especifica ha sido también incluida como ejemplo

y existe con el nombre IsodeAgent, y es la encargada de recibir las peticiones de asociación.

Un mecanismo similar al anterior es necesario para comunicarse con tecnologías de interfaz de usuario gráfica (GUI) las cuales tienen sus propios mecanismos de coordinación. En este caso, simplemente una nueva clase Coordinator especial es necesaria para cada uno de ellos. OSIMIS posee clases Coordinator especializadas para X-Windows Motif y TCL/TK, las cuales no trabajan para nuestro sistema operativo como ya habíamos anotado, pero su código fuente puede ser observado como un ejemplo en "/tesis/osimis/kernel" si se desea.

El como quedaría la herencia jerárquica para las clases nombradas anteriormente es mostrada en la Figura #21.

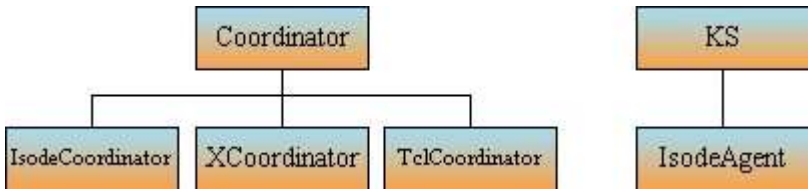


Figura #21 Clases Coordinator y KS Especificas

4.3 EL SISTEMA DE GESTION GENERICO (GMS).

El sistema de gestión genérico (GMS) provee el soporte para construir agentes que puedan ofrecer la completa funcionalidad del modelo de gestión OSI, incluyendo el "*scoping*" (alcance dentro del árbol de gestión, puede ser todo el árbol o ramas especificas), filtrado, control de acceso, respuestas enlazadas y "*cancel-get*". OSIMIS soporta completamente la gestión de objetos, el reporte de

eventos y el control "*log*", las funciones de gestión de sistemas (SMF), junto con el control de acceso y monitor métrico. En conjunción con el compilador GDMO el GMS ofrece una API de muy alto nivel para la integración de nuevas clases de objetos gestionados donde solo los aspectos semánticos (comportamiento) necesitan ser implementados. Este también ofrece diferentes métodos de acceso al recurso real asociado, incluyendo mecanismos "*proxy*", basados en el mecanismo de coordinación.

El sistema de gestión genérico es construido usando el mecanismo de coordinación y la infraestructura de soporte de alto nivel ASN.1 y muchas de sus facilidades son proporcionadas por tres clases C++:

- CMISAgent: La cual provee las facilidades al agente OSI.
- MO: Es una clase abstracta que provee el soporte al objeto de gestión genérico.
- MOClassInfo: Es una metaclase de una clase de objeto gestionado.

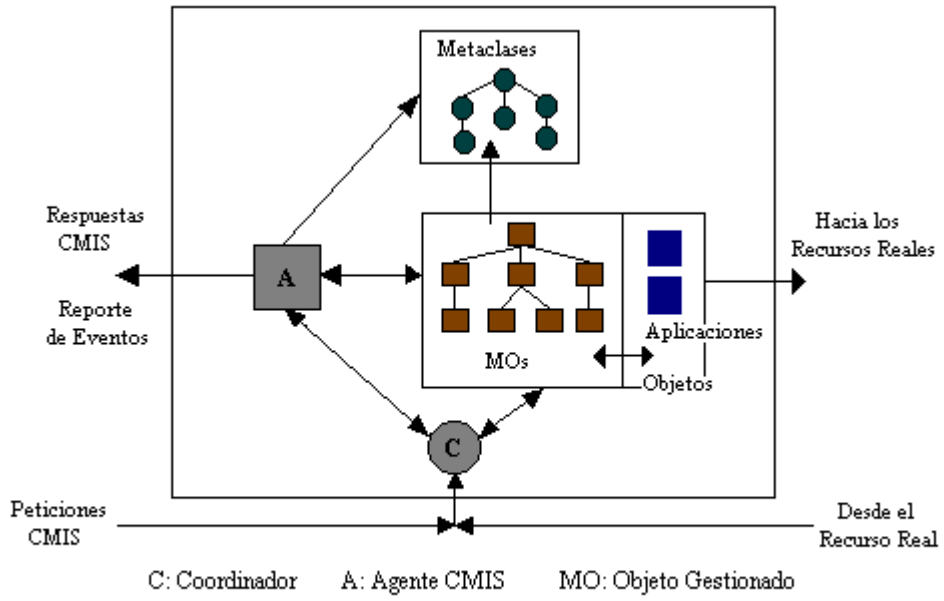


Figura #22 Arquitectura del GMS.

4.3.1 EL AGENTE CMIS.

La clase CMISAgent es una clase KS especializada, específicamente es un agente ISODE, que acepta asociaciones de gestión, siempre debe haber una única instancia de esta clase para cada aplicación que cumple el rol de agente. Su función es la de aceptar o rechazar asociaciones de acuerdo a la autenticación de la información, chequeando la validez de los parámetros de operación, encontrando el objeto base para la operación, aplicando *"scoping"* y filtrado, chequeando si la sincronización atómica puede ser implementada, chequeando los derechos de control de acceso y entonces aplicando la operación sobre el objeto(s) gestionado objetivo y retornando el resultado(s)/error(es). La llamada a un método debe siempre retornar un resultado, por ejemplo valores de atributos o un error, esto significa que los objetos gestionados cuya imagen es débilmente acoplada al recurso real y ejercen un régimen "acceso bajo petición externa" tendrán que acceder al recurso real de una forma sincrónica, lo cual resulta en el bloqueo de la aplicación hasta que el resultado es recibido. Esto es solo un problema si otra petición está esperando a ser servida o si muchos objetos son accedidos en una única petición a través de *"scoping"*, por lo tanto el uso de hilos de ejecución es una solución. Es de notar que la interfaz de CMISAgent a MO es bidireccional, así los objetos gestionados emiten notificaciones las cuales pueden ser convertidas a reportes de eventos y pasadas al gestor.

4.3.2 METACLASES E INSTANCIAS DE OBJETO GESTIONADO.

Cada clase de objeto gestionado específica necesita acceder a la información común a las clases que es independiente de todas las instancias y común a todas ellas. Esta información es perteneciente a atributos, acciones y notificaciones para la clase, valores iniciales por defecto de atributos, "plantillas" de objetos ASN.1 para manipular valores de acciones y notificaciones, etiquetas de enteros asociados a identificadores de objetos etc. Esto conduce a la introducción de una metaclase común para todas las clases de objetos gestionadas, esta es la

MOClassInfo. El árbol de herencia es representado internamente por instancias de esta clase enlazadas en un modelo de árbol como se muestra en la Figura #22 en el bloque metaclases.

Como veremos en la sección 4.5 (El Compilador GDMO) y en el capitulo siguiente, las clases específicas de objetos gestionados son realizadas creando clases C++ que se obtienen de compilar la MIB específica con el compilador GDMO y adicionándoles luego comportamiento, este comportamiento es implementado a través de un conjunto de métodos polimorfos de las clases C++ implementadas, los cuales pueden ser redefinidas para modelar el recurso real asociado. Las instancias del objeto gestionado son enlazadas internamente en un árbol reflejando las relaciones de contenimiento (ver la parte MOs de la Figura #22), de esta forma el *"scoping"* viene a ser una búsqueda en el árbol, mientras se tome especial cuidado al implementar las clases MO y en asegurar que el árbol refleje el estado del recurso real asociado antes de realizar el *"scoping"*, filtrado u otra operación de acceso.

4.3.3 ACCESO AL RECURSO REAL.

Algunos métodos deben ser proveídos para asegurar que los valores de los atributos recuperados en las peticiones GET de CMIS reflejen el verdadero estado de las cosas en el recurso real a ser gestionado, similarmente se debe asegurar que las peticiones SET de CMIS desempeñen realmente cambios en el estado del recurso real, por lo tanto la manera de comunicarse con el recurso real será altamente específica al recurso. El GMS apoya varios modelos de interacción entre el objeto gestionado y el recurso asociado. Estos pueden ser clasificados como:

- "Polling-(Sondeo)": En este el GMS llama un método del MO a intervalos, el cual estimula la comunicación con el recurso real. El polling puede ser más bien ineficiente puesto que grandes volúmenes de datos pueden ser

recuperados, los cuales nunca se necesitaran. Sin embargo, esta puede ser la única elección si las notificaciones están siendo emitidas cuando algún evento particular ocurre en el recurso real.

- “Acceso bajo una petición externa”: Aquí la comunicación con el recurso real es activada por una petición CMIS. Esto minimiza comunicación con el recurso real pero maximiza el actual proceso implicado en servir una petición. Es claro que allí no hay comunicación con el recurso real hasta que una petición externa es recibida. Esto significa que el método no puede ser usado para implementar eventos dependientes de datos tales como activar umbrales.

- “Activación por eventos”: El recurso real es quien inicia la comunicación (típicamente se invoca un mecanismo IPC (Interprocess Communication)). Esto es adecuado para configuraciones “asociadas débilmente” donde el recurso real es un proceso UNIX separado o incluso cuando está en una máquina apartada.

Usar diferentes mecanismos puede ser apropiado para atributos diferentes dentro del mismo MO. El GMS ofrece soporte para todos los métodos a través del mecanismo de coordinación. La organización interna de un agente basado en GMS en términos de la interacción con instancias de objetos es mostrada en la Figura #22, observemos que las instancias mostradas son solo las principales que definen el flujo interno de control, y también que la pila OSI es encapsulada por el objeto CMISAgent.

El GMS también contiene la implementación de los tipos de atributos "estándar" que deben cumplir los requerimientos para los tipos de atributos DMI básicos definidos en ISO 10165-2 (recomendación X.721 del UIT-T de 1992 [30]), estos tipos de atributos tienen una sintaxis que define rutinas para codificar y liberar estructuras C, decodificar los elementos de presentación de ISODE dentro de

estructuras C, imprimir estructuras C, analizar gramáticalmente, comparar dos instancias del mismo tipo, y rutinas para copiar instancias del mismo tipo. La implementación de esta sintaxis puede ser encontrada en "/tesis/osimis/agent/smi/SmiSntx.h" y "/tesis/osimis/agent/smi/SmiSntx.c".

Luego estas implementaciones de sintaxis son usadas en la definición de la clase C++ que implementara el tipo de atributo, todas las clases C++ deben ser derivadas de la clase base Attr (la cual puede ser consultada en el Anexo B), todos los tipos de atributos que implementa OSIMIS son listados a continuación en la Tabla #13A, y su definición puede ser observada en la recomendación citada anteriormente.

Tipos de Atributos Base		
AdditionalInformation	Integer	ObjIdList
AdministrativeState	IntegerList	OctetString
AlarmStatus	LogFullAction	OperationalState
AvailabilityStatus	MFilter	Real
Boolean	ManagementExtension	RealList
CThreshold	MgmtId	Replaceable
Counter	MgmtIdList	String
DestinationAddress	MScope	StringList
DName	Null	Time
DNList	ObjectInstance	TMark
Gthreshold	ObjectInstanceList	UsageState
Gauge	ObjId	

Tabla #13A Tipos de Atributos Base que Implementa OSIMIS

Además se definen tipos de atributos que han sido derivados desde otros, estos tipos de atributos son mostrados en la Tabla # 13B.

Tipo de Atributo	Tipo de Atributo Base
AttributeId	MgmtId
AttributeIdList	MgmtIdList
TideMark	TMark
TideMarkMin	TideMark
TideMarkMax	TideMark
SimpleNameType	String
PrintableString	String
PrintableStringList	StringList
GaugeInt	Gauge
GaugeReal	Gauge
GaugeThreshold	GThreshold
GaugeThresholdInt	GaugeThreshold
GaugeThresholdReal	GaugeThreshold
GraphicString	String
GraphicStringList	StringList
IA5String	String
IA5StringList	StringList
ObjectClass	MgmtId
ObjectClassList	MgmtIdList
CounterThreshold	CThreshold
AetStateArray	Array
EventType	MgmtId

Tabla # 13B Tipos de Atributos Derivados que Implementa OSIMIS

4.3.4 MONITOR METRICO.

Mientras el modelo agente-gestor no restringe la cantidad de inteligencia que puede ser incorporada y proveída por objetos gestionados, muchas especificaciones estándar se concentran en proveer un conjunto bastante rico de atributos y acciones que modelen la información y las interacciones posibles con los recursos subyacentes. Las notificaciones también son proveídas para reportes

significativos de eventos usualmente solo genéricos, por ejemplo creación de objetos, borrado de objetos y cambios del valor de atributos.

El grupo ISO/ITU-T reconoce la importancia de unas facilidades que permitan un monitoreo genérico y estandarizó las funciones de gestión de sistemas de resumen[1] y monitor métrico [31]. Haciendo tales funciones genéricas, es posible implementarlas y hacerlas parte de la infraestructura de la plataforma asociada.

Los objetos del monitor métrico pueden ser instanciados dentro de una aplicación en el rol de agente y ser configurados para monitorear a intervalos periódicos un atributo de un objeto gestionado del recurso real, el atributo puede pertenecer a un objeto gestionado del elemento de red, pero también a una aplicación de gestión de alto nivel siendo mapeada dentro de objetos gestionados de bajo nivel, a través de una función de conversión de información obviamente esta tiene que ser realizada para la aplicación especifica. El atributo observado debe ser del tipo Counter o Gauge [30] y el objeto métrico puede observar el valor del atributo directamente en su tipo o convirtiéndolo a una tasa sobre el tiempo (Gauge derivado). El refinamiento estadístico de los valores observados es también posible si se desea.

[1] Los objetos de resumen se refieren al "servicio que permite obtener un resumen de objetos en la gestión del sistema" que fue nombrado en el capitulo II como uno de los cuales no se logro hacer funcionar, y lo que hace este objeto es extender la idea de monitorear un atributo simple (como lo hace el monitor métrico) a monitorear múltiples atributos a través de un número de instancias del objeto gestionado seleccionado. Esto ofrece facilidades similares pero complementarias para objetos métricos, en este caso no hay comparaciones o umbrales, mas bien se usa el refinamiento estadístico y la obtención de resultados algorítmicos simples del valor observado (min, max, media y variación), y estos son reportados periódicamente al sistema de gestión interesado a través de notificaciones. El objeto gestionado observado y los atributos pueden ser especificados de cualquier forma, dando sus nombres explícitamente o a través del *"scoping"* y filtrado CMIS, los valores observados pueden ser datos "brutos". Estos pueden ser reportados de cualquier forma a cada periodo de observación o después de un número de periodos de observación (por eso la frase "resumen de objetos"). Para mas información se pude ver la recomendación X.739 del UIT-T [32].

La principal importancia de esta facilidad es el anexo de atributos de tipo GaugeThresholds y TideMarks al Gauge derivado resultante, el cual puede incrementar la calidad del servicio de alarmas e indicar la "*water mark*" alta y/o baja, como es deseado por sistemas que usan esta función. De hecho, el objeto métrico esencialmente realza el modelo de información "bruta" recibida del objeto observado. La funcionalidad del monitor métrico puede ser definida como:

- **Captura de datos:** A través de la observación o escaneo de un atributo del objeto gestionado.
- **Conversión de datos:** Permite convertir un tipo Counter o Gauge a un Gauge derivado.
- **Mejora de datos:** Permite el refinamiento estadístico del resultado derivado.
- **Generación de notificación:** Puede generar notificaciones de la QoS de alarmas y cambios en los valores de atributos.

El modelo de monitoreo métrico es mostrado en la Figura # 23.

El uso del monitor métrico proporciona gran flexibilidad en tiempo de ejecución, ya que una aplicación de gestión puede definir e instanciar un conjunto de objetos monitor para desempeñar estas provechosas actividades genéricamente sobre otros MOs existentes.

Básicamente un objeto monitor es usado para observar periódicamente un atributo que existe en otro objeto gestionado el cual es llamado " MO observado". El "atributo observado" dentro del MO observado debe ser del tipo Counter o Gauge, la información observada es almacenada internamente en el objeto monitor en un atributo local del tipo Gauge, luego el objeto monitor convierte internamente los atributos Counter observados a un porcentaje del tipo Gauge. Dependiendo del comportamiento que definamos para el objeto monitor, los umbrales internos pueden ser usados para determinar si una notificación "QoS de alarma" debe ser emitida cuando el atributo observado excede ciertos limites del umbral. Realizando

una petición CMIS se puede fijar o cambiar el limite del umbral usado en el objeto monitor, y turnar este on/off en tiempo de ejecución.

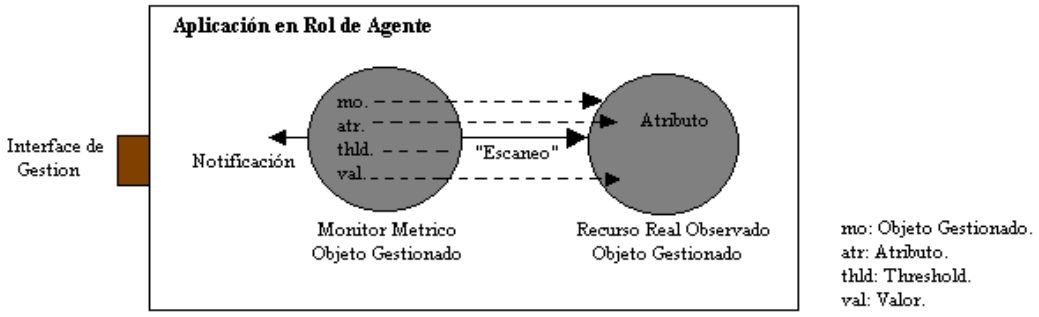


Figura #23 El Modelo del Monitor Métrico.

El uso de la MIB monitor métrico (por ejemplo en una función Q-adaptor (QAF)) provee a un gestor OSI gran flexibilidad. Por ejemplo las notificaciones no necesitan ser especificadas en la descripción de la MIB para todos los MOs definidos allí, en lugar de eso, en tiempo de ejecución la aplicación de gestión puede instanciar un objeto monitor apropiado para desempeñar esta función, esto desplaza más trabajo dentro de la aplicación de gestión, pero es un modelo usable genéricamente. Otro uso de los objetos monitor es que ellos pueden ser usados para eliminar alguna cantidad de trafico "*polling*" en la red anexada a un gestor OSI, cuando por ejemplo una interfaz M de TMN tal como SNMP es usada para comunicarse con el recurso real.

Cuando la comunicación con el recurso real usa una interfaz como SNMP es posible usar una pasarela genérica CMIP/SNMP, sin embargo, si no se usa el objeto monitor un gestor debe realizar "*polling*" sobre el agente SNMP para obtener información desde el recurso real. Por otro lado si hacemos uso del monitor métrico en la pasarela genérica, entonces todo el "*polling*" es aislado a la red local del agente SNMP, ya que el objeto monitor puede encargarse del

"*polling*" (por ejemplo escaneando internamente la pasarela) y solo emitir notificaciones al gestor cuando sea apropiado, esto ayuda a eliminar tráfico innecesario en la red de gestión, lo cual es una meta fundamental de un buen sistema de gestión de red. La Figura # 24 ayuda a aclarar estos conceptos.

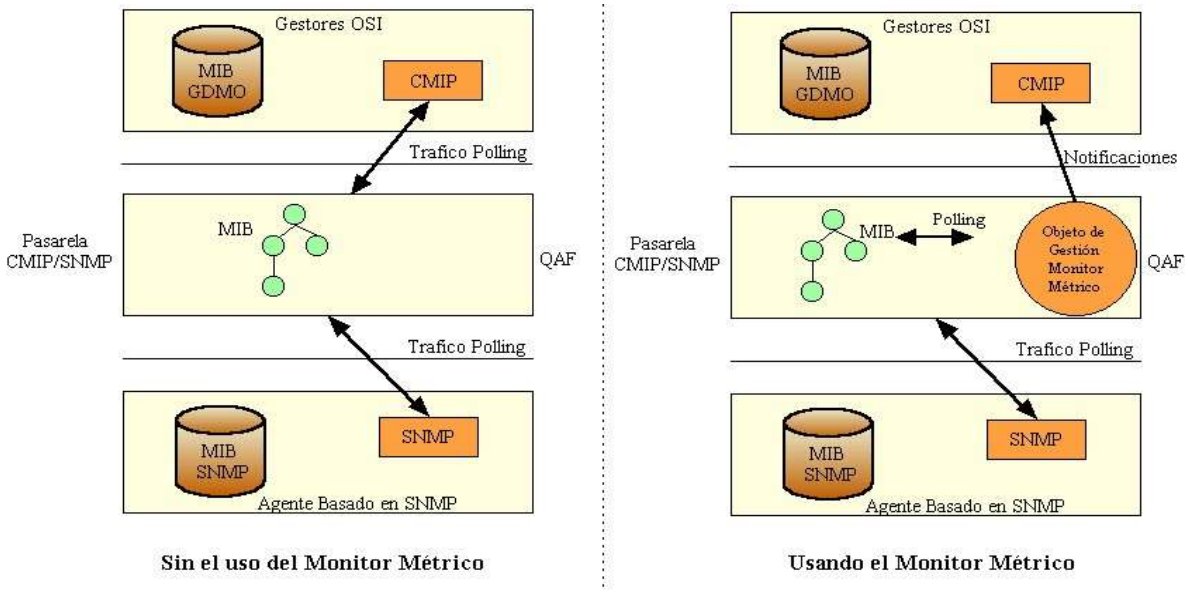


Figura # 24 Uso del Monitor Métrico

La clase de objeto gestionado objeto monitor esta definida en ISO10164-11 (recomendación X.734 del UIT-T [33]) y tiene la herencia jerárquica mostrada en la Figura # 25.

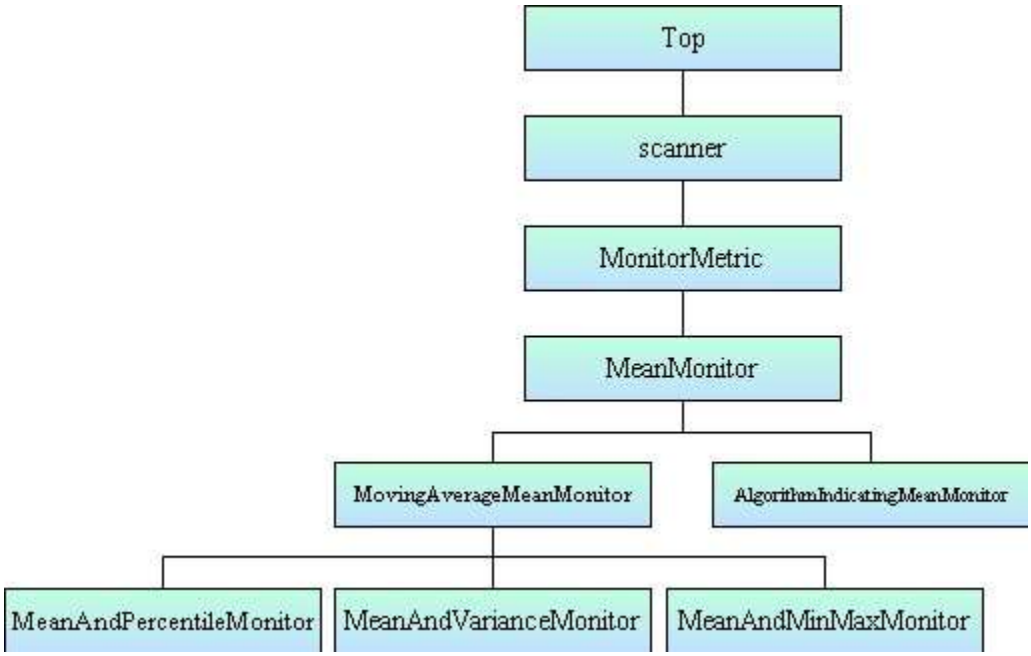


Figura # 25 Herencia Jerárquica del Objeto Monitor Metrico

Donde la clase base scanner define el comportamiento del scheduling/polling para escanear periódicamente los MO observados. La clase del objeto monitor MonitorMetric simplemente monitorea datos “brutos” desde los MOs observados. La clase derivada del objeto monitor MeanMonitor permite varios algoritmos de refinamiento (por ejemplo promedio variable estimado o promedio variable uniforme) para ser aplicados a los datos “brutos” medidos. Otras clases derivadas proveen características para conversión de datos tales como: media, media y variancia, media y porcentaje, media y mínimo/máximo.

4.4 LA MIB REMOTA (RMIB)

La naturaleza expresiva y orientada a objetos del modelo de gestión OSI [34] y el modelo de información asociado [35] proveen la infraestructura esencial a través de la cual las aplicaciones de gestión intercambian información de gestión usando el servicio CMIS. Aunque la estructura OSI de la información de gestión es

ampliamente basada sobre el diseño orientado a objetos y metodología de especificación, lo mismo no es verdad para el nivel del soporte de programación proveído por muchas APIs CMIS y este es el caso tanto para el MSAP de OSIMIS como para el XOM/XMP del X/Open.

Muchas realizaciones de APIs CMIS/P dejan al implementador encargarse del no placentero acceso a la información de gestión de mecanismos de bajo nivel, por ejemplo, a los identificadores de objetos usados por las clases, los atributos, nombres de reportes de eventos y acciones; mientras los nombres distinguidos en sus formas nativas son usados para direccionar instancias de objetos. Similarmente, los mecanismos de *"scoping"* y filtrado que son usados para seleccionar y eliminar los objetos deseados para la operación están sujetos a estructuras de datos de bajo nivel. Además, el protocolo CMIP es diseñado en tal forma que únicamente una instancia simple de información puede ser llevada en la PDU de respuesta. Por lo tanto, en el caso donde múltiples respuestas de objetos gestionados son retornados como resultado de una petición simple, la PDU de respuesta es enlazada con un identificador común y la tarea de ensamblar esas PDUs dentro de los resultados del objeto gestionado cae desafortunadamente en manos del implementador de la aplicación.

OSIMIS provee una API de acceso de alto nivel que permite un acceso eficiente a las bases de información de gestión remota, soportando a un alto nivel el desarrollo de aplicaciones en el rol de gestores, esta API es conocida como RMIB y ofrece una amistosa interfaz CMIS de alto nivel.

El objetivo primario detrás de la API RMIB es ocultar tanto como sea posible los detalles de bajo nivel de CMIS/P usando una abstracción orientada a objetos, mientras se mantiene su completo poder. Las ventajas obtenidas desde el punto de vista del implementador son dos: primero, el implementador es liberado de la manipulación de parámetros de gestión nada amistosos, y por lo tanto puede concentrarse en la realización de políticas de gestión, por ejemplo, la inteligencia

de la aplicación. Segundo, la reducción en el tamaño del código de la aplicación significa que el desarrollo puede ser más rápidamente culminado.

La API de acceso RMIB provee una abstracción orientada a objetos de alto nivel de las MIBs OSI remotas usando la noción de una “asociación de objeto”. Este objeto es usado para encapsular la asociación de gestión con un agente remoto, ocultando el uso de los parámetros CMIS/P de bajo nivel y accediendo el árbol de información de gestión (MIT) remoto. El procedimiento normal del control de asociación es ocultado a través de métodos amigables usando sólo el nombre de la aplicación y la información del host y posiblemente explotando el servicio de soporte de ubicación transparente. Hablando en términos de implementación, la asociación de objetos es representada en una clase C++ llamada *RMIBAgen*t, cada instancia de esta clase permitirá establecer una asociación con un agente remoto. Además para el control de asociación, la clase *RMIBAgen*t define un conjunto de mensajes de alto nivel en la API semejantes a CMIS, ocultando las actuales llamadas a CMIS/P. La Figura #26 muestra el modelo de interacción general. El nivel de soporte proveído por la clase *RMIBAgen*t habilita al implementador a pensar y diseñar la aplicación en términos de abstracciones, tomando en cuenta la clara separación respecto a los detalles del acceso a bajo nivel.

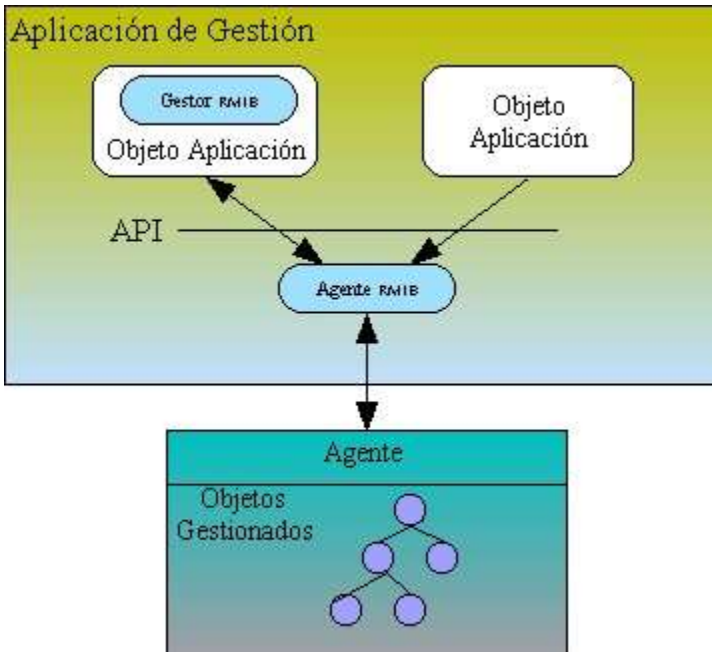


Figura #26 Interacción del Modelo de la MIB Remota

CMIS proporciona varias vías para seleccionar objetos gestionados para una operación, a través de un objeto gestionado base y también el tipo de subárbol a ser alcanzado bajo este. Un subárbol seleccionado de objetos gestionados puede estar sujeto a un criterio de selección, basado en las condiciones lógicas dadas en la expresión del filtro. El objeto gestionado base debe ser identificado a través de su nombre distinguido. El identificador de objeto de la clase de objeto gestionado puede también ser proveído para permitir acceso alomorfito al objeto. En la interfaz RMIB, los nombres de objeto gestionado son manejados a través de una notación basada en cadenas legibles al humano. Por ejemplo *"transportEntity"* y *"subsystemId=transport@entityId=isode"* se refieren respectivamente a la clase y nombre distinguido local del objeto transportEntity de ISO. Esta notación permitirá al programador hacer caso omiso de las complicadas estructuras de la API CMIS subyacente, dejando que el análisis necesario se efectué tras la interfaz. Usando la notación basada en cadena, los filtros pueden también ser expresados fácilmente y comprensivamente. Finalmente, los valores de atributos, acciones y

reportes de eventos usan la API ASN.1 de OSIMIS a través de las clases *Attr* (tipo general ASN.1) y *AVA* (declaración del valor de atributo).
El resultado retornado luego de una petición puede consistir de un número de objetos gestionados, así el objeto gestionado generado como resultado pude también contener declaraciones de valor de atributos tal como sucede con las primitivas de petición M-Get, M-Set, y M-Create. Varios métodos pueden ser usados para retornar los resultados desde el nivel de la interfaz al ambiente de aplicación, para lo cual hay que decodificar las respuestas enlazadas, el resultado puede ser entregado uno por uno o colectivamente en una unidad simple o contenedor, ambas aproximaciones son soportadas por RMIB, transfiriendo cada PDU respuesta dentro de una instancia de la clase *CMISObjec*t. Obviamente, la clase es bastante amplia para capturar la unión de parámetros de objetos gestionados que puedan ser retornados en todas las primitivas de respuesta, el resultado individual *CMISObjec*t[2] puede ser empaquetado dentro de una unidad de contenimiento simple, implementada por la clase *CMISObjectList*[2].

En cualquier diseño e implementación de sistemas distribuidos, es importante el permitir la máxima flexibilidad en términos de la interfaz de interacción, esto es, la interfaz debe operar en ambos modelos sincrónico y asincrónico. En el sincrónico RMIBAgent[2] opera como un mecanismo RPC en el cual la llamada es bloqueada mientras el resultado o error es recibido, el intervalo de tiempo muerto puede ser ajustado para que la llamada no este bloqueada demasiado tiempo, por otro lado, la total flexibilidad para realizar sistemas de operación de TMN complejos solo puede ser lograda con una interfaz asíncrona. Usando la idea de funciones "*call-back*", la API incluye una clase abstracta llamada *RMIBManage*r[2], esta clase provee un conjunto de métodos "*call-backs*" los cuales deben ser especializados

[2] Las clases RMIBAgent, RMIBManager, *CMISObjec*t y *CMISObjectList*, no serán profundizadas ni estudiadas posteriormente en el Anexo B como haremos con otras ya que la API RMIB es una unidad completamente desarrollada y no tiene ningún método que necesitemos conocer para crear correctamente un MO o un agente, ni tampoco método polimorfo alguno que podamos usar.

en una clase gestora asíncrona, y deben ser redefinidos con el comportamiento deseado para responder a los diferentes tipos de resultados CMIS. El RMIBAgent coloca en un buffer los resultados, para después decodificarlos y ensamblar las respuestas enlazadas, y finalmente devolver la llamada al RMIBManager.

El reporte de eventos es fundamental para la naturaleza manejada por eventos de la gestión OSI y es lograda a través de un objeto llamado eventForwardingDiscriminator (definido en la recomendación X.734). Una aplicación necesita manejar aquellos objetos explícitamente para pedir, terminar, suspender o resumir reportes. En el caso de la API RMIB, la manipulación explicita es ocultada tras la clase RMIBAgent, permitiendo la declaración del tipo de evento, el tiempo, la clase emisora y la instancia a través de un filtro arbitrario, todo en la forma de notación basada en cadena a través de métodos de alto nivel que oculten las operaciones CMIS necesarias para manipular el eventForwardingDiscriminator. Una ventaja de tal aproximación es que la proliferación de objetos eventForwardingDiscriminator en el MIT remoto es evitada a través del control centralizado, lo cual reduce el tamaño del agente remoto y permite una rápida evaluación del filtro. Las respuestas para confirmar el reporte de eventos y admitir la recepción es también manejada transparentemente por el RMIBAgent.
Como el reporte de eventos es inherentemente asíncrono, este necesita de la clase RMIBManager y sus mecanismos "*call-back*".

La infraestructura RMIB ofrece una facilidad de más alto nivel que la compleja API CMIS (MSAP), pero aún su naturaleza de transferencia de mensajes está estrechamente enlazada al CMIS, aunque esta facilidad es perfectamente adecuada aún para aplicaciones de gestión complejas ya que ofrece el completo poder de CMIS (*"scoping"*, filtrado, etc.), es deseable usar una facilidad que permita realizar prototipos rápidos de una aplicación gestora, tal facilidad fue

desarrollada por el proyecto RACE ICM [36] la cual es conocida como “Shadow MIB” (SMIB) o MIB imagen.

La MIB imagen (SMIB) ofrece la abstracción de los objetos gestionados en el espacio de dirección local, realizando la imagen del objeto gestionado real que es manejado por agentes remotos. Las ventajas de esta aproximación son dos:

- La API es menos compleja que CMIS, puesto que se accede a objetos locales y por tanto los nombres distinguidos, *“scoping”* etc., pueden ser reemplazados por punteros en espacio de dirección local.
- La existencia de los MOs como imágenes de objetos locales puede ser usada para mantener un cache de información y optimizar el acceso a los agentes remotos.

El modelo y las clases C++ para soportar la SMIB son muy similares a la misma RMIB, Este modelo es mostrado en la Figura # 27.

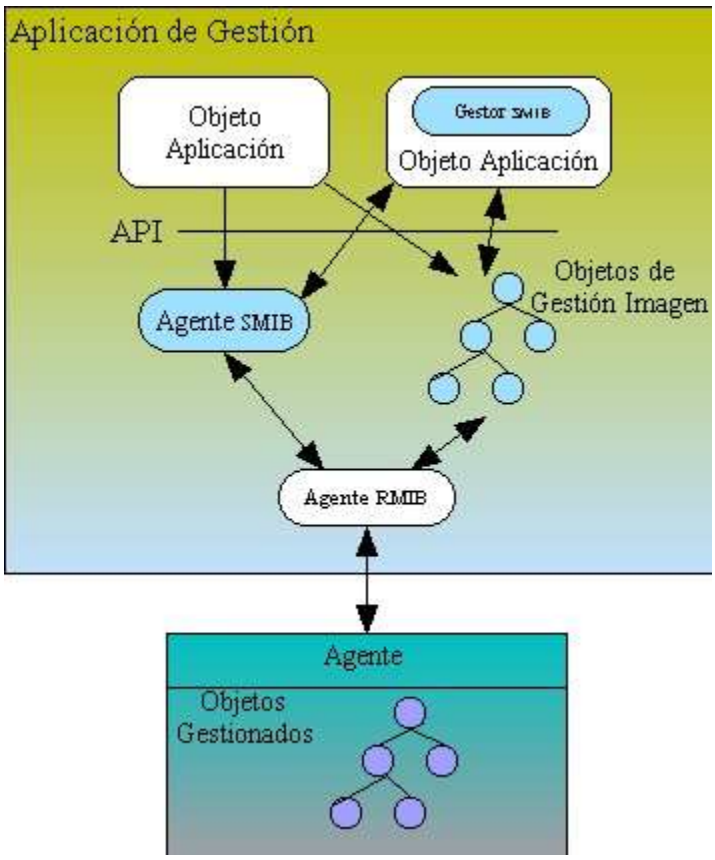


Figura # 27 Interacción del Modelo de la MIB Imagen

OSIMIS provee la infraestructura necesaria para anexar el mecanismo SMIB a este, y se espera poder anexarlo en un futuro, si se desea más información acerca de SMIB se puede acudir al documento: Pavlou. George, Tin Thurain, High-Level Access APIs in the OSIMIS TMN Platform: Harnessing and Hiding, 1994, el cual es proporcionado con el CD que acompaña la Monografía.

4.5 EL COMPILADOR GDMO.

Recordemos que la gestión de redes OSI usa un modelo orientado a objetos para describir dispositivos conectados a una red, cada dispositivo es representado como un conjunto de objetos, llamados objetos gestionados MOs; el nombre que

se da a un conjunto de MOs es base de información de gestión MIB. GDMO, el cual es un acrónimo para Guías para la Definición de Objetos Gestionados, provee una manera formal de describir los objetos, el GDMO es esencialmente un lenguaje orientado a objetos que se usa para definir un conjunto de clases de objetos gestionados (MOCs), los cuales en tiempo de ejecución son instanciados como MOs, para cada MOC, se especifica que posición tiene este en la jerarquía de herencia y contenimiento de una MIB, los nombres y tipos de sus atributos, las notificaciones que esta puede emitir, y las acciones que pueden ser desempeñadas por esta.

Un compilador GDMO debe ser capaz de tomar una MIB que se ha hecho pública y producir un agente de gestión de red que implemente la MIB. Pero en la práctica, esto no es tan simple, por dos razones principalmente:

- Un agente de gestión de red requiere una gran cantidad de infraestructura software, ya que cualquier plataforma de gestión de redes OSI debe soportar el protocolo CMIP, el cual es usado para transferir información entre agentes y gestores. Por lo tanto una implementación de GDMO que provea una plataforma de software para la construcción de agentes de gestión de red necesita de alguna aplicación que se encargue de realizar el trabajo de codificar y decodificar paquetes que pasan hacia y desde un agente, manejar todas las peticiones CMIP, coordinar todos los eventos de comunicación y timers y gestionar la MIB; en OSIMIS el encargado de realizar estas tareas es el GMS. Pero un compilador GDMO para OSIMIS tendrá que generar MOs conformes a los archivos cabecera y métodos de OSIMIS ya que cada MOC en OSIMIS es representado por una clase C++, la cual en tiempo de ejecución es instanciada como un MO.

- Un agente también tiene que contener código que interconecte los MOs con el recurso real que es gestionado; este código tiene que ser escrito a mano

para que extraiga información desde el recurso real y almacene esta en los atributos de los MOs. Por ejemplo, en el caso de la MIB Internet, el agente tiene que hacer llamadas al Kernel de UNIX a fin de extraer información acerca del estado de las conexiones TCP/IP en una estación de trabajo. El compilador GDMO por consiguiente tiene que ser capaz de asociar código generado y código escrito a mano.

Por las razones anteriormente expuestas, OSIMIS provee un compilador GDMO que para producir un agente de gestión de red trabaja en conjunción con el GMS y permite la inclusión de código escrito a mano.

El uso de este compilador es importante y será materia de estudio en el Capitulo siguiente.

4.6 SOPORTE PARA ASN.1 DE ALTO NIVEL.

OSIMIS provee un mecanismo para soportar la manipulación ASN.1, que protege al programador de los detalles y le permite realizar una programación distribuida usando objetos C++ como tipos de datos, esto se logra usando polimorfismo para encapsular el comportamiento en los tipos de datos, y así establecer como la codificación y decodificación deben ser desempeñadas a través de un compilador ASN.1, el cual produce clases C++ para cada tipo. Los métodos de codificación, decodificación, análisis gramatical, impresión, comparación y obtención del próximo elemento son producidos automáticamente por el compilador. Además un constructor muy importante del lenguaje ASN.1 el “ANY DEFINED BY” es soportado por el compilador a través de bases de datos (realizadas como tablas en archivos) que permiten mapear los tipos a la sintaxis. Esta aproximación ASN.1 es usada por los servicios OSIMIS de alto nivel tales como GMS, RMIB y SMIB.

El compilador ASN.1 de OSIMIS nos permitirá incluir nuevos tipos de atributos a los ya definidos, y trabajando junto al compilador GDMO podemos adicionar estos a los objetos gestionados. El uso de este compilador se podrá ver en el Capitulo siguiente.

4.7 LA PASARELA GENERICA CMIP/SNMP.

Para soportar la arquitectura TMN, un dispositivo necesita una interfaz Q3; en el ambiente de gestión de red OSI el cual emplea TMN, un gestor y agente se comunican entre si usando una interfaz Q3 y CMIP.
Comparando con el ambiente Internet, donde el protocolo es SNMP, la gestión de red OSI provee un poderoso modelo de gestión orientado a objetos, mientras la gestión de red SNMP sigue una menos poderosa pero simple aproximación que ya esta ampliamente difundida, las dos tecnologías pueden coexistir ya que el nivel de aportación de la tecnología de gestión SNMP es alta, por tanto el mejor camino para esta coexistencia es proveer una vista OSI de todos los recursos SNMP gestionables existentes. Hay una amplia cantidad de productos de gestión existente que no usan TMN, pero TMN provee una solución para enfrentar este problema usando el QAF (Q Adaptor Function), adaptando el protocolo existente ofrecido a la interfaz Q de TMN.

A través del proyecto ICM se desarrollo un QAF o pasarela genérica entre SNMP/CMIP conocido como el IQA (Internet Q-Adaptor), el cual fue adicionado en OSIMIS, ofreciendo la capacidad de hablar directamente a agentes SNMP desde una estación de gestión de red OSI. Esta pasarela presenta una interfaz Qx al gestor CMIP y ha sido implementada basándose en los estándares dados por el NMF.

El IQA permite a una plataforma de gestión conforme a TMN gestionar uno o más dispositivos de gestión SNMP, cuando el IQA esta en funcionamiento toma

peticiones CMIP y mapea estas dentro de peticiones SNMP, las respuestas SNMP y los "*traps*" son trasladados dentro de respuestas y eventos CMIP, así el QAF aparece como un agente CMIP al gestor CMIP y un gestor SNMP al agente SNMP. Esto es mostrado en la Figura # 28.

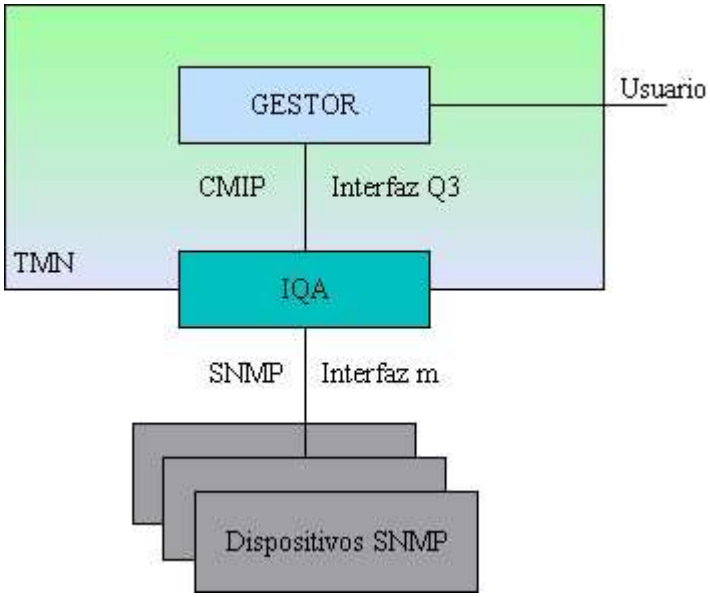


Figura # 28 IQA (Adaptador Q de Internet).

Lo que hace el IQA es trasladar un resultado M-Get de CMIP en un Get SNMP e igualmente un resultado M-Set CMIP en un set SNMP. Similarmente, los "*traps*" SNMP son convertidos dentro de M-EventReports de CMIP. Los M-Create y M-Delete de CMIP pueden ser aplicados sólo a tablas SNMP, por lo tanto cualquiera de estas operaciones puede devolver un error CMIS, las funciones más sofisticadas como las de configuración requerirán aplicaciones especificas en el gestor CMIP.

El gestor CMIP puede gestionar varios agentes SNMP, los cuales tienen unas MIBs SNMP propietarias diferentes, la IQA dependiendo del gestor CMIP puede residir en la misma plataforma o puede ser un dispositivo aislado.

La estrategia de usar una pasarela adiciona manejabilidad a un mundo completo de recursos con capacidades de gestión SNMP, pero el uso del modelo OSI esta restringido a la semántica del modelo SNMP, ya que el modelo SNMP es un subconjunto del modelo OSI con ciertos comandos del lado OSI del IQA que no tienen sus equivalentes en el lado SNMP.

En OSIMIS para lograr suministrar una interfaz Qx a un dispositivo SNMP gestionable, se debe trasladar la MIB SNMP dentro de una MIB GDMO usando un compilador conocido como imibtool [37] (desarrollado por Jim Reilly de VTT Finlandia, y facilitado por este para incluirlo en OSIMIS), el cual se encarga de trasladar el SMI al GDMO, luego se debe realizar algún procesamiento manual para acudir al compilador GDMO, que producirá los archivos necesarios para que el IQA pueda incluir esta nueva MIB SNMP. Un esquema del proceso de construcción es mostrado en la Figura #29.

Figura #29 Proceso General de Conversión de MIBs SNMP a GDMO

En el Anexo A podemos observar más detalladamente cómo trasladar MIBs de SNMP a GDMO usando imibtool y el compilador GDMO para poderlas anexar al IQA y así realizar la gestión de sus objetos desde OSIMIS.

4.8 SISTEMAS LOGGING Y LA GESTION DE ESTADOS Y OBJETOS.

4.8.1 SISTEMAS LOGGING.

OSIMIS contiene una implementación de la función de control Log que es descrita en ISO 10164-6 (recomendación X.735 de la ITU-T [38]), la cual es realizada por 3 objetos C++ que han sido implementados, y poseen la herencia jerárquica mostrada en la Figura # 30:

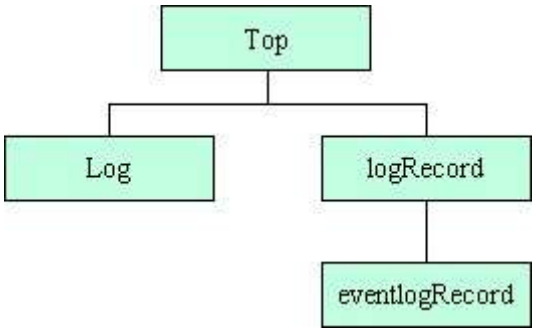


Figura # 30 Herencia Jerarquica del Sistema Logging.

Log, LogRecord y evenlogRecord son conformes a los descritos en ISO 10164-6, ISO 10165-2 e ISO 10165-2 respectivamente.

Esta implementación actual del control Log en el GMS se enfoca en proveer un mecanismo que permita el almacenamiento de información de una manera no volátil, por lo tanto usa un sistema de logging basado en disco, así los registros son almacenados en disco usando una versión modificada de la interfaz de la base de datos gdbm [39] desde GNU. Este reemplaza el ndbm obviando así las restricciones de almacenamiento que este impone. El código gdbm fue modificado de manera tal que el archivo es solo asegurado ("*locked*") mientras es accedido (el comportamiento gdbm por defecto es mantener el archivo asegurado mientras esta abierto), esta modificación fue hecha para permitir accesos simultáneos al archivo desde varios recursos.

4.8.1.1 Almacenamiento de los registros log en el disco: Los archivos en el disco son mantenidos en el directorio “/osimis/logs” que seleccionamos durante la configuración e instalación de OSIMIS en el archivo “/tesis/osimis/etc/isotailor”. Un nombre único para cada archivo log es derivado desde el nombre del agente y el identificador (logId) del objeto Log en la siguiente forma:

<nombre agente>-<nombre NSAP>-log<logId>

por ejemplo para un objeto Log con "logId=1" el cual es parte de un proceso agente con nombre “SMA”, ejecutándose en el host “osimis.ucauca.edu.co” tendríamos:

SMA-osimis-log1

Cuando un evento es logged, este es almacenado en este archivo. Cuando un objeto log es involucrado en una operación delete de CMIS, su entrada correspondiente es entonces removida desde este archivo.

4.8.1.2 Formato de los registros log : El formato de registro consiste de 8 campos presentados en la Tabla # 14, conteniendo caracteres ASCII, imprimibles y alfanuméricos, y con cada campo delimitado por un carácter “:” así:

eventLogRecord:2:1021017194024Z:discriminator:discriminatorId=ds:objectCreation:1021017194024Z:30550a01013150300e8005590302074180055903020303300e8005590302073f06055903020601300b8005590302070119026473300980055903020738a900300a8005590302071f0a0101300a800559030207230a0101

Campo	Descripción
0	nombre del tipo de registro log
1	LogRecordId
2	versión formateado del tiempo de logging
3	nombre de la clase de objeto que emite la notificación
4	nombre de la instancia del objeto que emite la notificación
5	nombre de la notificación
6	versión formateada del tiempo de notificación
7	representación hexadecimal del dato del evento codificado en BER

Tabla # 14 Descripción de Cada Campo del Registro Log.

Como podemos observar la forma en que se almacenan los logs no es muy agradable al lector humano, por lo tanto para hacer esta información más comprensible OSIMIS provee un ejecutable llamado “readlog”, el cual desplegará más amistosamente la anterior información, tal como muestra la Figura # 31.

```
[root@osimis logs]#readlog SMA-osimis-log1

objectClass:eventLogRecord
logRecordId:2
loggingTime:1021017194024Z
managedObjectClass:discriminator
managedObjectInstance:discriminatorId=ds
eventType:objectCreation
eventTime:1021017194024Z
additionalText:ObjectInfo: SourceIndicator: managementOperation
{
objectClass:  discriminator
nameBinding:  discriminator-system
discriminatorId:  ds
discriminatorConstruct:  (NULL)
administrativeState:  unlocked
operationalState:  enabled
}
```

Figura # 31 Despliegue del Registro Log Usando readlog.

Usar readlog es muy sencillo como podemos ver en la Figura #31, sólo requiere como argumento el nombre del log a leer, si se desea más información sobre este ejecutable se pueden ver las ayudas en línea de OSIMIS (paginas del manual en línea usando el comando “man”).

4.8.1.3 Almacenamiento de los registros mib en el disco: Existe otro archivo que se almacena en disco llamado “mib” el cual es solo otro archivo gdbm, pero que contiene la información sobre el objeto gestionado. que se ubica en el mismo directorio “/osimis/logs” que seleccionamos durante la configuración e instalación de OSIMIS en el archivo ”/tesis/osimis/etc/isotailor”. Un nombre único para este archivo es derivado desde el nombre del agente de la siguiente forma:
<nombre agente>-<nombre NSAP>-mib

Por ejemplo para el archivo mib del agente “SMA” sobre el host “osimis.ucauca.edu.co” será almacenado como:

SMA-osimis-mib

4.8.14 Formato de los registros mib: El formato de registro consiste de una variable con un número de campos conteniendo caracteres ASCII, imprimibles, y alfanuméricos, con cada campo delimitado con un carácter “:” así:

Log:logId=1:11:objectClass=-:nameBinding=-:logId=020101:
discriminatorConstruct=a900:administrativeState=0a0101:operationalState=0a01
01:availabilityStatus=3100:logFullAction=020100:maxLogSize=020100:currentLog
Size=020100:numberOfRecords=020100:

La Tabla # 15 describe los campos anteriores, y el número exacto de campos en el registro depende del valor en el campo 2, cada uno de los campos después del campo dos tiene el formato:

<nombre del atributo>=<representación hexadecimal del valor codificado en BER>

si un valor de atributo fue almacenado, un carácter “-“ es mostrado en su lugar, así:

objectClass=-:

tal como podemos observar en el ejemplo anterior.

Campo	Descripción
0	nombre de la clase de objeto
1	nombre de la instancia de objeto
2	lista de atributos

Tabla #15 Descripción de Cada Campo del Registro mib

De igual forma que se puede usar readlog para observar el registro log, lo podemos usar para el registro mib como muestra la Figura # 32.

```
[root@osimis logs]#readlog SMA-osimis-mib

Managed Object class:Log
Managed Object instance:{loId=1}

objectClass:log
nameBinding:log-system
logId=1
discriminatorConstruct:(NULL)
administrativeState:unlocked
operationalState:enabled
avaibalityStatus:{}
LogFullAction:wrap
maxLogSize:0
currentLogSize:585
numberOfRecords:26

--Press RETURN –

1 entries in SMA-osimis-mib

[root@osimis logs]#
```

Figura #32 Despliegue del Registro mib Usando readlog.

4.8.1.5 Reactivando logs: Cuando un agente es reiniciado, este automáticamente restaurara los objetos Log que existían antes que el agente cayera. Esto se logra utilizando la información almacenada acerca de los objetos log y eventlogRecord en el disco.

Cuando un Log es creado por medio de CMIS, este es almacenado también en los archivos, cualquier cambio en el objeto se repite en los archivos y así el objeto siempre será reiniciado desde su estado más reciente. Cuando un objeto Log es envuelto en una operación delete de CMIS, su correspondiente entrada es removida desde los archivos, y también su correspondiente archivo log es destruido. Sin embargo, la actual implementación ofrece la posibilidad de renombrar o borrar los archivos log.

Por defecto los archivos log son renombrados, por ejemplo si un eventLogRecord con logId=1 fue detectado por medio de CMIS, su entrada debe ser removida desde el archivo mib y el archivo log:

SMA-osimis-log1
Es renombrado a:

SMA-osimis-log1.old

Para habilitar el borrado de estos archivos (en lugar del renombrado), se debe colocar en comentarios la línea mostrada en la Figura #33 perteneciente al archivo Log.inc.hcl del GMS ubicado en "/tesis/osimis/agent/gms", recompilando posteriormente el agente correspondiente, en este caso SMA.

```
...⇒⇒⇒⇒⇒ Línea #1
#include "GenericKS.h"

/*
** The default action is to rename the gdbm files when a log is deleted.
** Comment out the next line to enable deleting of log files.
*/
#define LOG_RenameOldLogFiles          → COMENTAR ESTA LINEA

#define LOG_Directory     "logs"
...
```

Figura # 33 Edición del Archivo Log.inc.hcl.

4.8.2 LA GESTION DE OBJETOS Y ESTADOS EN OSIMIS.

Las funciones de gestión de objetos y de estado (recomendación X.730 [40]), junto con las notificaciones de violación de seguridad, el reporte de alarmas y el QoS de alarmas, son fundamentales para proveer el poder de la gestión OSI a través del reporte de eventos. La gestión de objetos define tres notificaciones de importancia fundamental estas son:

* objectCreation
* objectDeletion
* attributeValueChange

La gestión de estados también define unos pocos atributos de estado genéricos, suministrando un modelo para transiciones entre estos estados y suministrando además una importante notificación:
* stateChange

OSIMIS ya implementa la sintaxis para aquellas notificaciones y además para los más importantes de los atributos de estado:

* operationalState
* administrativeState
* usageState
* availabilityStatus
* alarmStatus

El modelo que utiliza OSIMIS para automatizar la emisión de todas las anteriores notificaciones es usar un método llamado triggerEvent para accionar una notificación y un método buildReport que debe ser suministrado por el implementador del MO, ambos métodos están definidos dentro de la clase Top

(ver el Anexo B). El modelo es bastante general así que cuando en el futuro un mecanismo de notificación diferente sea implementado, la API propuesta no cambiara.

Este modelo nos permitirá por lo tanto tomar un control automático de las notificaciones cuando un MO está siendo creado, borrado o modificado a través de la interfaz CMIS, debido a que es transparente para los implementadores y no tienen que preocuparse por estas, además el modelo nos provee un manejo amistoso para accionar aquellas notificaciones que se necesiten debido a la actividad en el recurso subyacente, es decir el implementador solo debe realizar el método buildReport para el objeto especifico, como veremos en el Capitulo siguiente.

CAPITULO V

AGENTES Y OBJETOS GESTIONADOS DE OSIMIS

Ahora que se ha compilado e instalado OSIMIS, y se conoce su infraestructura interna se puede iniciar el estudio de su uso como plataforma de gestión analizando los objetos gestionados y agentes que implementa OSIMIS. Al mismo tiempo y talvez más importante para nosotros, se estudiara la manera en que se puede conjugar la compilación e instalación de OSIMIS con el conocimiento de su infraestructura e implementación interna, para encausarlos en el desarrollo de nuestros propios objetos gestionados que representen y permitan la comunicación con recursos reales o aplicaciones, así como también de nuestros propios agentes de gestión que manejarán estos objetos.

Pero antes de embarcarnos en estas discusiones, estudiaremos más a fondo una herramienta que ya conocemos y que será fundamental para lograr nuestro objetivo, esta herramienta es el compilador GDMO con que cuenta OSIMIS.

5.1 EL COMPILADOR GDMO DE OSIMIS.

Luego de la correcta instalación de OSIMIS, el compilador GDMO a quedado listo para ser usado, y queda ubicado dentro de la ruta “/osimis/bin” donde se sabe

están ubicados los binarios de OSIMIS, el nombre del ejecutable del compilador es "gdmo-cmpl" y antes de poder utilizarlo debemos aclarar que este usa varias bases de datos y "*scripts*" de Unix para realizar sus tareas de compilación, que deben ser accedidas en tiempo de ejecución, estas bases de datos y "*shells*" se encuentran ubicados en la ruta "/osimis/etc/gdmodir" y se necesita que la variable de ambiente OSIMISETCPATH esté bien configurada como se vera más adelante, de lo contrario se recibirá diferentes errores al tratar de usar el compilador.

5.1.1 EJECUTANDO EL COMPILADOR.

El ejecutable gdmo-cmpl toma como argumento el nombre de un módulo GDMO, para así generar las clases de objetos de gestión C++ compatibles con OSIMIS. El compilador es actualmente invocado dos veces, primero para generar una lista de la sintaxis referenciada en el módulo GDMO y segundo para generar el código C++ del objeto gestionado.

Si algún error es encontrado en el módulo GDMO, el compilador imprimirá un mensaje de error y gdmo-cmpl terminará. Si no se encuentran errores, los archivos .h y .cc para todos los objetos gestionados y clases de atributos serán generados junto con un archivo makefile llamado Makefile.mo (esto será detallado en la sección 5.1.4). Cuando compilemos los objetos gestionados, debemos asegurar también que el compilador C++ pueda encontrar el directorio donde se encuentran los archivos de cabecera de OSIMIS. Una forma de asegurar el correcto ambiente de funcionamiento del compilador es usando las convenciones que usa OSIMIS, de tal forma que en lugar de llamar directamente la utilidad make de Unix, podemos crear un "*shell script*" llamado make que lea al CONFIG.make en el directorio principal de OSIMIS (sin olvidar ejecutar "chmod +x make" para hacer el "*script*" ejecutable). Un ejemplo de un "*script*" de OSIMIS es mostrado en la Figura #34.

```
#!/bin/sh
exec/bin/make TOP:    /osimis –f  ../CONFIG.make  -f Makefile $ {1+"$@"}
```

Figura # 34 Ejemplo de un "*script*" make de OSIMIS

5.1.2 BASES DE DATOS.

El compilador GDMO para realizar su tarea lee tres bases de datos, una base de datos de objetos gestionados (mo.dat), una base de datos de atributo (attr.dat), y una base de datos de sintaxis (syntax.dat). Las tres bases de datos son archivos ASCII que tienen tres campos, el primero es el nombre de un MO, el tipo de atributo o sintaxis del atributo, el segundo es el nombre de la clase C++ correspondiente que implementa esta y el tercero es el nombre del archivo cabecera que contiene la declaración de la clase C++.

El compilador GDMO usa las tres bases de datos en la siguiente forma:

- **Base de datos MO:** Cuando un MO es referenciado en la oración DERIVED FROM en la respectiva plantilla MO, el compilador usa esta base de datos para resolver cualquier referencia externa (por ejemplo para MOs que no se definen en el módulo GDMO actual). Si el compilador no encuentra una entrada en la base de datos, generará un error, la base de datos por defecto contiene los MO's que son implementados en OSIMIS (ejemplo top y system). La Figura #35 muestra la base de datos MO por defecto.

```
#####################################################################
#
#       mo.dat is a database of External MO classes
#       The fields of each record (line) in the file are:
#
# 1) GDMO MO name 2) MO C++ Class Name 3) MO C++ Class Header file
#
#
#####################################################################
top                 Top                     Top.h
```

```
system          System          System.h
managedElementmanagedElementmanagedElement.h
```

Figura #35 Base de Datos MO

- **Base de datos de atributo:** Cuando un atributo es referenciado en la oración DERIVED FROM de una plantilla de atributo, el compilador GDMO usa esta base de datos para resolver cualquier referencia externa (por ejemplo para atributos que no están definidos en el módulo GDMO actual). Si el compilador no encuentra una entrada en la base de datos, generará un error, la base de datos por defecto contiene los siguientes atributos estándar de OSIMIS mostrados en la Figura #36.

```
#####################################################################
#
#       attr.dat is a database of attribute classes
#       The fields of each record (line) in the file are:
#
# 1) GDMO Attribute Type Name 2) Attribute C++ Class Name 3) C++ Header file
#####################################################################
#
# These are the generic attribute types from which other may be derived
# using the DERIVED FROM GDMO clause
#
counter                 Counter                 Counter.h
counterThreshold                CounterThreshold                CounterThld.h
counter-Threshold               CounterThreshold                CounterThld.h
gauge                   Gauge                   Gauge.h
gaugeThreshold          GaugeThreshold          GaugeThld.h
gauge-Threshold         GaugeThreshold          GaugeThld.h
tideMark                        TideMark                        TideMark.h
tide-Mark                       TideMark                        TideMark.h
member                  DNList                  DNList.h
#
# These are commonly used attributes from the DMI document
# (ISO 10165-2 / CCITT X.721) which may be referred to from other GDMO specs
#
objectClass             ObjectClass             ObjectClass.h
managedObjectClass              ObjectClass             ObjectClass.h
managedObjectInstance   ObjectInstance          ObjectInstance.h
eventType                       EventType               EventType.h
discriminatorConstruct  MFilter                 Filter.h
destination                     DestinationAddressGms           Destination.h
administrativeState             AdministrativeState             AdministrativeState.h
operationalState        OperationalState                OperationalState.h
usageState                      UsageState              UsageState.h
availabilityStatus      AvailabilityStatus              AvailabilityStatus.h
```

Figura #36 Base de Datos de Atributo.

- **Base de datos de sintaxis:** Esta es una base de datos de sintaxis ASN.1. El compilador GDMO usa esta base de datos para resolver que código debe generar cuando encuentra la oración WITH ** SYNTAX (donde ** es ATTRIBUTE, INFORMATION etc.) en el módulo GDMO. Cuando se ejecuta gdmo-cmpl, una nueva base de datos sintaxis es generada la cual es una combinación de la base de datos sintaxis por defecto y una lista de la sintaxis referenciada en el módulo GDMO. La Figura #37 lista la sintaxis estándar de OSIMIS. Al comienzo de la tabla, el lenguaje ASN.1 para la sintaxis implementada en los tipos OSIMIS, es listada en comentarios (denotada por un carácter # al comienzo de la línea). Vale la pena notar que si la sintaxis OSIMIS por defecto es lo único que necesitamos, podemos ejecutar el compilador GDMO referenciado en la sintaxis OSIMIS. Los atributos genéricos desde los cuales un atributo puede ser derivado es algunas veces referenciado en la oración WITH ATTRIBUTE SYNTAX de una plantilla atributo. La principal cuestión que una plantilla de atributo adiciona a una sintaxis es el registro (matriculación).

```
#################################################################
#
#       syntax.dat is a database of syntax classes
#       The fields of each record (line) in the file are:
#
# 1) GDMO Syntax Name 2) C++ Class Name 3) C++ Header file
#
#
# INTEGER                   Integer                 Integer.h
# REAL                      Real                    Real.h
# OCTET STRING                      OctetString                 OctetString.h
# IA5String                 String                  IA5String.h
# SET OF GraphicString      StringList                        StringList.h
# SET OF IA5String                  IA5StringList               IA5StringList.h
# UTCTime                   Time                    Time.h
# CMISFilter                MFilter                 Filter.h
# Scope                             MScope                              Scope.h
# OBJECT IDENTIFIER         ObjId                   ObjId.h
# SET OF OBJECT IDENTIFIER  ObjIdList                       ObjIdList.h
# ObjectClass               ObjectClass             ObjectClass.h
# SET OF ObjectClass        ObjectClassList         ObjectClassList.h
#
#################################################################
#
# The following should not be used if the DERIVED FROM clause is used
```

```
# for the generic attribute types [ counter, gauge, counter-Threshold
# gauge-Threshold and tide-Mark ] instead of the WITH ATTRIBUTE SYNTAX.
# In this case, the attr.dat file contains the right entries.
# In case though that this is not the case, the syntax of those
# attributes is included below.
#
Count                   Counter                 Counter.h
CounterThreshold        CounterThreshold        CounterThld.h
ObservedValue           Gauge                   Gauge.h
GaugeThreshold          GaugeThreshold          GaugeThld.h
TideMarkInfo            TideMark                TdeMark.h
#
# The following are OSIMIS-implemented syntaxes that may be re-used
#
SimpleNameType          SimpleNameType          SimpleNameType.h
Boolean                 Boolean                 Bool.h
Integer                 Integer                 Integer.h
Real                    Real                    Real.h
OctetString             OctetString             OctetString.h
GraphicString           GraphicString           GraphicString.h
IA5String               IA5String               IA5String.h
PrintableString         PrintableString         PrintableString.h
IntegerList             IntegerList             IntegerList.h
RealList                RealList                RealList.h
GraphicStringList       GraphicStringList       GraphicStringList.h
IA5StringList           IA5StringList           IA5StringList.h
PrintableStringList     PrintableStringList     PrintableStringList.h
Time                    Time                    Time.h
UTCTime                 Time                    Time.h
GeneralizedTime         Time                    Time.h
CmisScope               MScope                  Scope.h
CMISScope               MScope                  Scope.h
Scope                   MScope                  Scope.h
CmisFilter              MFilter                 Filter.h
CMISFilter              MFilter                 Filter.h
Filter                  MFilter                 Filter.h
ObjectIdentifier        ObjId                   ObjId.h
ObjectIdentifierList    ObjIdList               ObjIdList.h
ObjectClass             ObjectClass             ObjectClass.h
ObjectClassList         ObjectClassList         ObjectClassList.h
ObjectInstance          ObjectInstance          ObjectInstance.h
ObjectInstanceList      ObjectInstanceList      ObjectInstanceList.h
DistinguishedName       DName                   DName.h
EventTypeId             EventType               EventType.h
AttributeId             AttributeId             AttributeId.h
#
# The following are syntaxes for attributes that may be re-used
#
DiscriminatorConstruct  MFilter                 Filter.h
Destination             DestinationAddress      Destination.h
AdministrativeState     AdministrativeState     AdministrativeState.h
OperationalState        OperationalState        OperationalState.h
UsageState              UsageState              UsageState.h
AvailabilityStatus      AvailabilityStatus      AvailabilityStatus.h
```

Figura #37 Base de Datos de Sintaxis

5.1.3 SCRIPTS.

Como se había mencionado anteriormente, el compilador gdmo-cmpl hace uso de varios "*scripts*" que se encuentran en "/osimis/etc/gdmodir", estos son mostrados en la Tabla #16.

Script	Descripción
Start.gen	Este es el principal "*script*" usado para la generación de código, por defecto hace uso de otros dos "*scripts*" que se encuentran en el mismo directorio estos son: header.hg y methods.ccg. El "*script"* start.gen genera las cabeceras y archivos método para cada MO; si se desea proveer "*scripts*" a la medida para generar código para una MO, start.gen usara estos en lugar de los "*scripts*" por defecto. Los nombres de los "*scripts*" de generación de código de cabecera y métodos hechos a la medida son formados por la concatenación del nombre del MO con ".hg" y ".ccg" respectivamente.
Dumpsyntax.gen	Este es un "*script*" compilador encargado de extraer toda la sintaxis referenciada en el modulo GDMO.
Init.gen	Este "*script*" crea las inicializaciones de clases y atributos referenciadas en el modulo GDMO.
Makefile.gen	Es el encargado de generar el archivo Makefile.mo que permitirá generar la librería para los MOs.
Oidtable.gen	Este es usado para ayudar a automatizar la generación de las tablas de identificadores de objetos oidtable.at y oidtable.gen, el nombre del archivo de salida es oidtable.tmp y es desde el cual podemos obtener los valores apropiados para oidtable.at y oidtable.gen.

Tabla #16 Scripts Usados por el Compilador GDMO.

Los "*scripts*" pueden ser modificados para cambiar el comportamiento del compilador si se desea, en nuestro caso no es necesario y utilizamos los "*scripts*" por defecto que proporciona OSIMIS.

5.1.4 ARCHIVOS GENERADOS.

Al usar el compilador y ejecutarse este sin encontrar errores, generara un archivo cabecera y un archivo método para cada MO definido en el módulo GDMO. Los nombres de los archivos cabecera y métodos son formados concatenando el nombre del MO con ".h" y ".cc" respectivamente. También generara un Makefile llamado Makefile.mo para ayudar en la compilación del código fuente escrito a mano, y un archivo llamado oidtable.tmp el cual contiene una lista de las cosas (atributos, clases, etc) que debemos registrar en el módulo GDMO, el contenido de este archivo puede ser cortado y pegado dentro de los oidtables relevantes.

5.1.5 OPCIONES DEL COMPILADOR GDMO.

Es posible usar varias opciones para el compilador GDMO, estas son mostradas en la Tabla #17.

Opción	Significado	Descripción
-d	debug	Esta opción ejecuta el compilador en modo depurar. Un menú de depuración es desplegado, el cual tiene varias opciones que nos permitirán desplazarnos por las jerarquías de herencias y contenimiento, los objetos gestionados y la estructura de tablas de símbolo internas al compilador.
-t	TopMO (por defecto es top)	Especifica cual es el objeto gestionado Top.
-m	MOdatabaseName (por defecto es mo.dat)	Esta opción especifica donde encontramos la base de datos de los MOs externos, de esta forma podemos usar

		una base de datos diferente a mo.dat.
-a	AtributeDatabaseName (por defecto es attr.dat)	Esta opción especifica donde encontramos la base de datos de los atributos GDMO externos.
-s	SyntaxDatabaseName (por defecto es syntax.dat)	Esta opción especifica en que base de datos está la sintaxis ASN.1.
-x	ScriptName (por defecto es start.gen)	Esta opción nos permite especificar que "*script*" de generación de código usará el compilador, gdmo-cmpl usa esta opción para ejecutar el "*script*" que generara la lista de sintaxis referenciada en un modulo GDMO.

Tabla #17 Opciones del Compilador GDMO.

5.1.6 INCLUYENDO CODIGO.

El código escrito a mano debe ser combinado con el código generado por el compilador gdmo-cmpl, por lo tanto hay ciertas convenciones que se adoptan en los "*scripts*" por defecto que controlan la generación de código de los compiladores, de tal forma que es posible cambiar estas convenciones relativamente fácil, modificando los "*scripts*".

Para cada clase de objeto gestionado, el gdmo-cmpl busca o espera 3 archivos de código escrito a mano para ser combinados con el código generado por el compilador. No es necesario crear estos archivos si un MO no requiere la definición de comportamiento especifico alguno. Los archivos escritos a mano son incluidos usando la directiva "#include" del pre-procesador. El compilador establece las dependencias relevantes en el Makefile así que si deseamos cambiar alguno de los archivos escritos a mano, solamente las clases C++ de MOs deben ser recompiladas. Usando la directiva "#include" se generara código algo desagradable, sin embargo la alternativa de copiar código escrito a mano dentro de los archivos generados es menos atractiva; por lo tanto si realizamos modificaciones en uno de los archivos escritos a mano, debemos ejecutar

nuevamente el compilador GDMO, de esta manera los cambios estarán incorporados dentro de los archivos C++ de MOs generados.

Dos archivos son incluidos dentro del archivo cabecera generado por el compilador. Un archivo, cuyo nombre es formado por la concatenación del nombre de MO con ".inc.h" el cual debe contener datos definidos por el usuario y declaraciones de métodos; estas declaraciones son insertadas dentro de la definición de clases generadas por el compilador. El formato del archivo ".inc.h" debe ser una etiqueta seguida por declaraciones, donde la etiqueta es cualquiera public:, private:, o protected:, recordemos que una declaración de clases C++ puede por ejemplo tener más de una etiqueta Private:.

El nombre del otro archivo incluido dentro del archivo cabecera es formado por la concatenación del nombre del MO con ".inc.hcl". El propósito del archivo es permitir al usuario proveer un definición de clases (la parte cl de sufijo es por class), cuando una clase MO necesita ser heredada de otra clase C++, un ejemplo de esto podría ser cuando una clase MO tiene que heredar desde la clase KS de OSIMIS. Cuando el compilador GDMO halla un archivo ".inc.hcl", este no suministra la definición de la clase a sí mismo, solamente el cuerpo de la definición de la clase, prescindiendo así de tener la definición de la clase al final del archivo.

Un tercer archivo es incluido dentro de los archivos método generados por el compilador, el nombre de este archivo es formado por la concatenación del nombre del MO con ".inc.cc", el archivo debe contener métodos definidos por el usuario para la clase MO. El compilador GDMO no hace otra suposición acerca del formato de los archivos ".inc.cc".

Resumiendo, el código escrito a mano y el generado por el compilador es combinado usando un número de convenciones que tienen que ser programadas dentro de los "*scripts*" de generación de código de OSIMIS. El lenguaje "*script*"

esta diseñado no solamente para afrontar el problema de generación de código para diferentes plataformas de gestión de red sino también para proveer un ambiente flexible para combinar código escrito a mano.

5.1.7 ENTENDIENDO EL FUNCIONAMIENTO.

La Figura #38 ilustra la interacción de todos los items ("*scripts*", bases de datos etc.) que están involucrados con el compilador GDMO. Los rectángulos redondeados indican programas, mientras los rectángulos normales indican archivos que son generados o leídos como entradas.

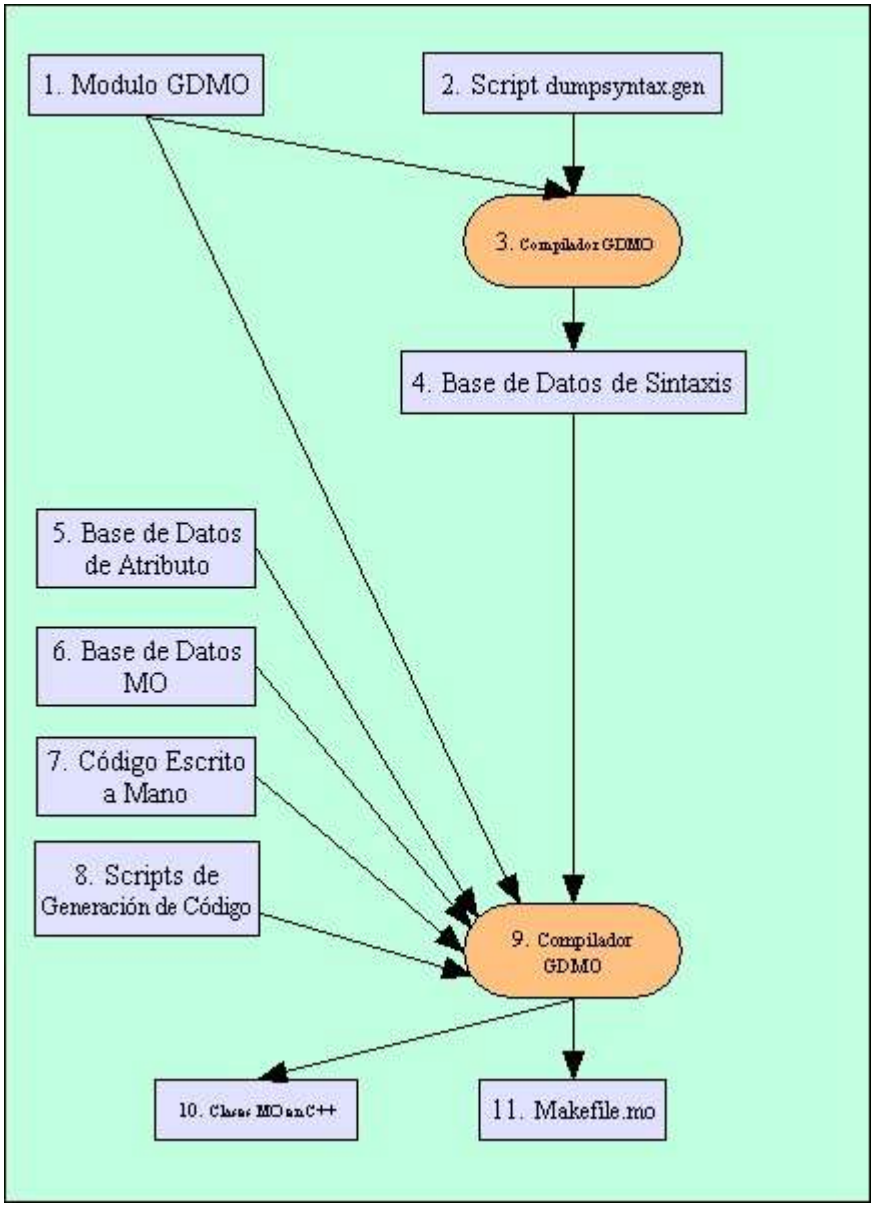


Figura #38 Usando el Compilador GDMO.

1. Modulo GDMO
2. El script compilador dumpsyntax.gen que extrae toda la sintaxis referenciada en el módulo GDMO.
3. Compilador GDMO, manejado por dumpsyntax.gen (2), extrae la sintaxis usada en el módulo GDMO (1) y crea una base de datos de sintaxis referenciada en el módulo GDMO (4).

4. Base de datos de sintaxis referenciada en el módulo GDMO generada por el script dumpsyntax.gen (2) y el compilador GDMO (3).
5. Base de datos de atributo.
6. Base de datos de objetos Gestionados.
7. Código escrito a mano.
8. Scripts de generación de código (donde están start.gen, makefile.gen, init.gen, oidtable.gen).
9. El compilador GDMO es ahora llamado por segunda vez. Este es manejado por los "*scripts*" de generación de código (8), y toma como entradas las 2 bases de datos (4, 6) y genera clases C++ de MO conforme a OSIMIS (10) y un Makefile (11).
10. Clases C++ (Archivos .c y .h) de MO generados por (9).
11. Makefile.mo (script) generado por (9).

5.2 OBJETOS GESTIONADOS QUE IMPLEMENTA OSIMIS.

En este apartado estudiaremos todos los objetos gestionados que hacen parte de OSIMIS y los cuales podemos utilizar luego de su correcta instalación, nosotros como usuarios de OSIMIS podemos hacer uso de tres agentes de gestión que son SMA, IQA y ODP; el como usar estos tres agentes será caso de estudio posteriormente, por ahora nos interesa saber que ellos utilizan diferentes objetos gestionados tanto genéricos como específicos que les permiten realizar sus diversas tareas.

La implementación de estos objetos gestionados se inicia con la definición de la MIB en el formato GDMO que los contiene, agregándole luego comportamiento con la introducción de código (lo cual veremos posteriormente). El extenso código fuente que implementa estos objetos puede ser observado y estudiado como ejemplo para futuras implementaciones de objetos, y se encuentra en las siguientes rutas:

“/osimis/agent/gms”
“/osimis/agent/isode_mib”
“/osimis/agent/monmet_mib”
“/osimis/examples/odp”
“/osimis/agent/ux_mib”
“/osimis/proxy/iqa/proxy”

Pero por ahora nuestra intención aquí es estudiar y entender el comportamiento de estos objetos gestionados, es decir, conocer que nos permite gestionar cada objeto, que recurso representa, que atributos tiene y cuales podemos manipular, que acciones podemos efectuar, que notificaciones puede generar, etc. Todo esto debe ser definido en la MIB que contiene el objeto, además el comportamiento lo define el código fuente particular para cada uno de ellos. Las MIBs también se encuentran definidas en las rutas anteriores, en cada una de estas rutas existe un archivo de nombre “MIB.gdmo” definiendo los objetos, por lo tanto daremos un nombre distintivo a cada una de estas MIBs dependiendo de los objetos que implementan, estos nombres son: MIBgms, MIBisode, MIBmonmet, MIBodp, MIBux, y MIBiqa.

Las seis MIBs que se nombraron anteriormente se pueden separar debido a los objetos que definen en MIBs genéricas que son : MIBgms, MIBmonmet, y MIBiqa y en MIBs especificas que son MIBisode, MIBodp, y MIBux; debido a su importancia todas estas MIBs son presentadas en el Anexo C, y es muy recomendable leerlo, sobra decir que es necesario un conocimiento del lenguaje GDMO para comprenderlo.

Para lograr explicar entonces el papel de las clases de objetos gestionados debemos hacer uso de la definición que se da de estas en la MIB correspondiente y del comportamiento que se les asigna mediante código en C++, por esta razón

para cada MIB hemos realizado figuras y tablas explicativas que resumen la definición en formato GDMO y el comportamiento adicionado mediante código en C++, aclarando así sus clases de objetos, herencia jerárquica, atributos, acciones, notificaciones, y comportamiento.

5.2.1 MIBs DEFINIENDO LOS OBJETOS GESTIONADOS.

Antes de iniciar la explicación de las clases de objetos, aclaramos que hay ciertos atributos y notificaciones definidas genéricamente que se refieren a la "notificación de objetos y estados de OSIMIS" los cuales ya estudiamos en el capitulo anterior, la Tabla #18 nos recuerda estos.

Atributos	Notificaciones
• OperationalState • administrativeState • usageState • availabilityStatus • alarmStatus	• objectCreation • objectDeletion • attributeValueChange • stateChange • qualityofServiceAlarm

Tabla #18 Atributos y Notificaciones Genéricos de OSIMIS.

Además existe un atributo especial que tiene el formato "<nombre de variable>Id", por ejemplo scannerId, el cuales una cadena arbitraria que se usa para identificar siempre una instancia de la clase especifica por medio de su nombre distinguido.

Ahora si entraremos en detalle al estudio de las clases de objetos gestionados, debemos recordar eso si, que cada atributo corresponde a un tipo especifico de los definidos por el GMS, los cuales ya fueron listados (Boolean, Counter, Gauge, Integer, IntegerList etc), además cada atributo permite diferentes operaciones como GET, SET, GET-REPLACE, REPLACE-WITH-DEFAULT, las cuales son definidas en GDMO.

5.2.1.1 MIBgms: La MIBgms define unas clase de objetos genéricos, los cuales permiten utilizar características básicas del modelo de gestión OSI, estos objetos obviamente están definidos en la Recomendación X.721-ISO/IEC 10165-2: 1992 y son genéricos ya que prestan la abstracción necesaria para el manejo del reporte de eventos y control Log. Además esta MIB define el objeto base necesario para formar un árbol de gestión el cual es conocido como system, por lo tanto cualquier agente de OSIMIS está obligado a utilizar el objeto base definido, de la misma forma si se desean los beneficios del reporte de eventos y control Log, estos objetos también deben ser definidos para el agente.

La Figura #39 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

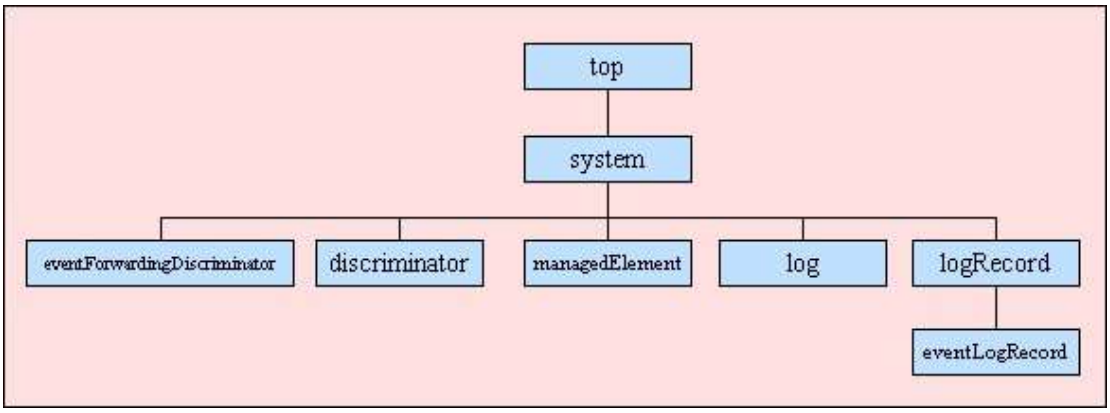


Figura # 39 Herencia jerárquica de los Objetos en la MIBgms.

Objeto system:	
La clase de objeto gestionado system es de uso obligatorio en todos los agentes puesto que el estándar especifica que el objeto Top en cada sistema gestionado será una instancia de la clase system, la cual puede tener cualquier RDN.	
Atributos	**Descripción**

1. systemId 2. systemTitle 3.operationalState 4. usageState	[2]2. Contiene el nombre global con su dominio

Tabla # 19A Objeto system.

Objeto managedElement:	
Esta clase es necesaria para complementar la vinculación de nombres tal como se define en la recomendación M.3100 de 1995, y esta implementación es conforme a esta.	
Atributos	**Descripción**
1. managedElementId 2. systemTitle 3. alarmStatus 4. administrativeState 5. operationalState 6. usageState	2. Contiene el nombre distinguido de esta clase u objeto

Tabla # 19B Objeto managedElement.

Objeto discriminator:	
La clase de objeto gestionado discriminator es la que se utiliza para definir los criterios que controlaran los servicios de gestión, es decir esta es la encargada de seleccionar los reportes de eventos que van a ser enviados a través de los valores de sus atributos. Por medio de discriminator podemos recibir cualquier reporte especifico o recibir todos los reportes de eventos desde la MIB remota cuando ninguna expresión filtro es especificada.	
Atributos	**Descripción**
1. discriminatorId 2. discriminatorConstruct 3. administrativeState 4. operationalState	2. Es el atributo que nos permite definir que eventos van a ser reportados, representándolos como una expresión filtro.
Notificaciones	
1. stateChange	

[2] El numeral presente en la descripción corresponde al número del atributo.

2. attributeValueChange
3. objectCreation
4. objectDeletion

Tabla # 19C Objeto discriminator.

Objeto eventForwardingDiscriminator:	
Esta clase es derivada desde discriminator y el sistema gestor necesita este objeto de soporte de gestión especial en el sistema de gestión remoto a fin de estar habilitado para recibir el reporte de eventos.	
Atributos	**Descripción**
1. destination	1. Especifica la máquina a la cual se van a enviar los reportes.

Tabla # 19D Objeto eventForwardingDiscriminator.

Objeto Log:	
Se encarga de definir los criterios para controlar la acción de registro de información en el sistema.	
Atributos	**Descripción**
1. logId 2. discriminatorConstruct 3. administrativeState 4. operationalState 5. availabilityStatus 6. logFullAction 7. maxLogSize 8. currentLogSize 9. numberOfRecords	6. Se usa para avisar cuando el fichero del registro cronológico Log esta lleno. 7. Especifica el tamaño máximo del log (el valor 0 es usado para no colocar restricciones de tamaño). 8. Especifica el tamaño actual del log. 9. Especifica la cantidad de logs actuales.
Notificaciones	
1. stateChange 2. attributeValueChange 3. objectCreation 4. objectDeletion	

Tabla # 19E Objeto Log

Objeto logRecord:	
Es una clase abstracta necesaria para derivar eventLogRecord y que nos permite definir los registros contenidos en un fichero log (registro cronológico) de objetos gestionados.	
Atributos	**Descripción**
1. logRecordId 2. loggingTime	2. es el tiempo en que se realiza el log.

Tabla # 19F Objeto logRecord

Objeto eventLogRecord:	
Es el objeto que nos permite definir la información almacenada en el fichero log como resultado de la recepción de notificaciones o informes de eventos.	
Atributos	**Descripción**
1. managedObjectClass	1. Especifica la clase de objeto que genera el log.
2. managedObjectInstance	2. El nombre de la instancia de la clase anterior.
3. eventType	3. El tipo de evento que genero el log.
4. eventTime	4. El tiempo en que se genero el evento.
5. additionalText	5. Contiene la información sobre el objeto que genero el log, con sus valores de atributo encriptado usando BER.

Tabla # 19G Objeto eventLogRecord

5.2.1.2 MIBux: La MIBux define un par de objetos específicos que nos permitirán gestionar el número de usuarios que se encuentran en un sistema en particular, permitiéndonos además dos formas de comunicación con el recurso real, estas son usando el “polling” o una petición externa.

La Figura #40 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

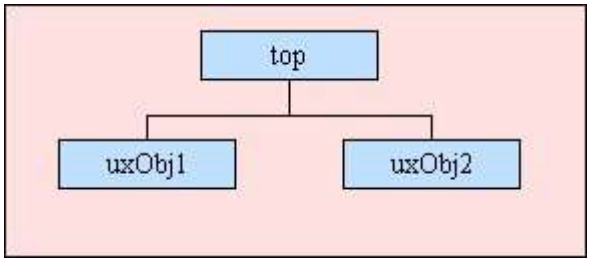


Figura # 40 Herencia jerárquica de los Objetos en la MIBux.

Objeto uxObj1:	
Este objeto es el que nos permite gestionar el número de usuarios en el sistema, además de definir una acción que nos permite recuperar los nombres de los usuarios activos en el sistema.	
Atributos	**Descripción**
1. uxObjId 2. sysTime 3. wiseSaying 4. nUsers	2. El tiempo actual de acuerdo al reloj del sistema. 3. Es una cadena arbitraria la cual puede ser fijada usando el servicio de gestión OSI. 4. Un atributo de tipo Gauge que refleja el número actual de usuarios logged en el sistema.
Acciones	**Descripción**
1. getUserNames	1. La acción que nos permite obtener el nombre de los usuarios.
Notificaciones	
1. objectCreation 2. objectDeletion 3. attributeValueChange	

Tabla # 20A Objeto uxObj1.

Objeto uxObj2:	
Este objeto nos permite relacionar umbrales y “TideMark” (limites extremos) con el atributo nUsers del objeto uxObj1, se ha creado un objeto aparte para esta tarea con el fin de demostrar que la funcionalidad de este objeto puede ser reemplazado usando el monitor métrico, mostrando así la gran ventaja en haber definido en OSIMIS un objeto monitor métrico genérico.	
Atributos	**Descripción**
1. nUsersThld	1. Este es un atributo del tipo Gauge-Threshold que permite

2. nUsersTideMark	manejar umbrales del atributo nUsers. 2. Este es un atributo del tipo TideMark que nos permite manejar el limite extremo del atributo nUsers.
Notificaciones	
1.qualityofServiceAlarm	

Tabla # 20B Objeto uxObj2.

5.2.1.3 MIBmonmet: La MIBmonmet define un conjunto de clases de objetos genéricos que le otorgan a cualquier agente que los use la posibilidad de trabajar con las funciones de gestión estándar que define el modelo monitor métrico que fue tratado en el Capitulo IV. Esta MIB implementa los objetos siguiendo la Recomendación X.721-ISO/IEC 10164-11 con excepción de unos pocos atributos contenidos que se implementaron como complemento.

La Figura #41 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

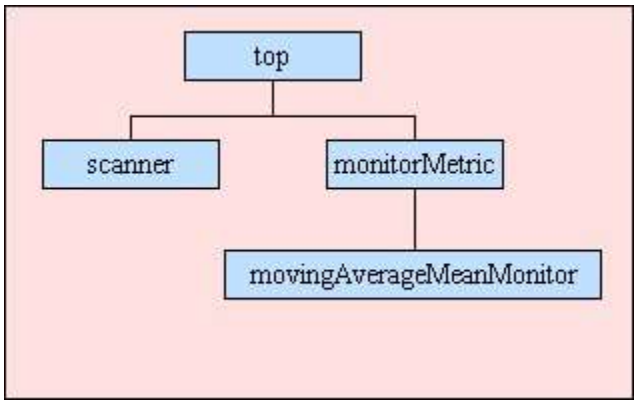


Figura # 41 Herencia jerárquica de los Objetos en la MIBmonmet.

Objeto scanner:	
Esta clase nos permite definir el comportamiento del "scheduling" y "polling" para poder escanear periódicamente los MO observados.	
Atributos	**Descripción**
1. scannerId 2. administrativeState 3. granularityPeriod	3. Es el tiempo entre dos "*scan*" sucesivos, y está dado en segundos.

4. operationalState	
Notificaciones	
1. objectCreation 2. objectDeletion 3. attributeValueChange 4. stateChange	

Tabla # 21A Objeto scanner.

Objeto monitorMetric:	
Este objeto es el que se encarga de monitorear los datos "brutos" desde los MO's observados.	
Atributos	**Descripción**
1. observedObjectInstance	1. Identifica el objeto que se observará.
2. observedAttributeId	2. Identifica el atributo del objeto a observar.
3. derivedGauge	3. Es un atributo del tipo Gauge derivado desde el valor del atributo.
4. severityIndicatingGaugeThreshold	4. Es un umbral relacionado con el atributo derivedGauge.
5. severityIndicatingTideMarkMax 6. severityIndicatingTideMarkMin	5.-6. Estos dos atributos no están definidos en el estándar, son adiciones que se le hacen al objeto para poder ver los valores máximo y mínimo del TideMark asociado al atributo observado, de esta manera obtenemos información adicional.
7. previousScanCounterValue	7. Valor del atributo observedAttributeId en el "scan" previo.
8. previousScanGaugeValue	8. Valor del atributo derivedGauge en el "scan" previo.
9. counterOrGaugeDifference	9. Este atributo no es estándar y la razón para ello es que OSIMIS aun no soporta apropiadamente los paquetes counterDifferencePackage y gaugeDifferencePackage definidos en la recomendación, los cuales son usados para proveer información adicional. Si el valor proveído para este atributo en la inicialización es TRUE, el Gauge derivado contendrá la diferencia entre el umbral y el Counter o Gauge derivado, si no este tiene el mismo valor observado. Por defecto el valor inicial es FALSE.

Notificaciones
1. qualityofServiceAlarm

Tabla # 21B Objeto monitorMetric.

Objeto movingAverageMeanMonitor:	
Este objeto es el que permite realizar varios algoritmos de refinamiento sobre los datos obtenidos por el objeto monitorMetric, como por ejemplo el promedio variable estimado o el promedio variable uniforme.	
Atributos	**Descripción**
1. estimateOfMean 2. movingTimePeriod 3.estimateOfMeanSeverityIndicatingGaugeThreshold 4. estimateOfMeanSeverityIndicatingTideMarkMax 5. estimateOfMeanSeverityIndicatingTideMarkMin	1. Es la media de los valores observados del atributo. 2. nos permite saber y definir cada cuanto se calcula la media. 3. Es la media del atributo severityIndicatingGaugeThreshold. 4.-5. Estos dos atributos no están en el estándar, son adiciones que permiten observar la media de los atributos severityIndicatingTideMarkMax y severityIndicatingTideMarkMin

Tabla # 21C Objeto movingAverageMeanMonitor.

5.2.1.4 MIBisode: La MIBisode define unas clases de objetos específicos que fueron concebidos por los realizadores de OSIMIS para que estos nos permitieran realizar una gestión de la capa de transporte que implementa la plataforma ISODE, por lo tanto podemos gestionar las aplicaciones que utilizan a ISODE como lo son FTAM y el programa de prueba imisc. Este objeto nos permite comunicación con el recurso real por medio de peticiones externas y por el método de activación por eventos.

La Figura #42 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

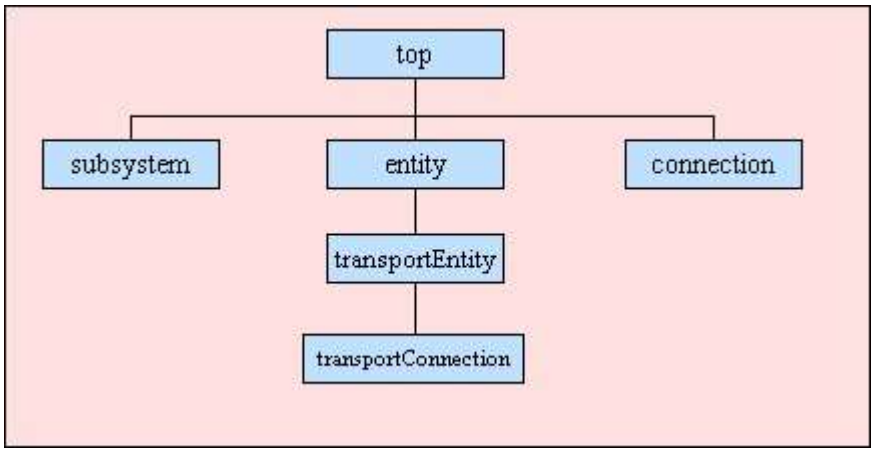


Figura # 42 Herencia jerárquica de los Objetos en la MIBisode.

Objeto subsystem:	
Este es el objeto utilizado para realizar la correcta vinculación de nombres en esta MIB, es necesario que exista para poder utilizar los demás objetos definidos en esta MIB, aunque no presta otro servicio más que el ya mencionado.	
Atributos	**Descripción**
1. subsystemId	

Tabla # 22A Objeto subsystem.

Objeto entity:	
Esta es una clase de objeto abstracta que permite a su clase derivada transportEntity modificar el estado operacional de la gestión de la capa de transporte e identificar su ubicación.	
Atributos	**Descripción**
1. entityId 2. operationalState 3. localSapName	3. Es el nombre del punto de acceso al servicio local, por ejemplo localhost.localdomain.
Notificaciones	
1. objectCreation 2. objectDeletion	

Tabla # 22B Objeto entity.

Objeto connection:	
Esta es una clase de objeto abstracta que permite a su clase derivada transportConnection identificar las ubicaciones locales y remotas.	
Atributos	**Descripción**
1. connectionId 2. localSapAddress 3. remoteSapAddress 4. creationTime	2. Dirección SAP local. 3. Dirección SAP remota. 4. Tiempo de creación de la conexión.
Notificaciones	
1. objectCreation 2. objectDeletion	

Tabla # 22C Objeto connection.

Objeto transportEntity:	
Es el objeto que nos permite ver diferentes parámetros del estado actual de la capa de transporte, y se encarga de generar los objetos connection y transportConnection necesarios para gestionar cada conexión.	
Atributos	**Descripción**
1. openConnections 2. previousConnections 3. localSuccessfulConnections 4. remoteSuccessfulConnections 5. localUnsuccessfulConnections 6. remoteUnsuccessfulConnections 7. localCongestionErrors 8. localConfigurationErrors 9. localChecksumErrors 10. totalNumberTpduSent 11. totalNumberTpduReceived 12. totalNumberTpduResent	Todos estos atributos nos permiten conocer el estado actual de las conexiones que se encuentran operando en la capa de transporte que implementa la plataforma ISODE, y todos ellos están definidos acorde a la recomendación X.721 (1992) del CCITT.

Tabla # 22D Objeto transportEntity.

Objeto transportConnection:	
Este objeto es creado por transportEntity cuando una conexión es abierta y se destruye automáticamente cuando es terminada.	
Atributos	**Descripción**
1. protocolClass 2. octetsSent 3. octetsReceived 4. octetsResent 5. pdusSent 6. pdusReceived 7. pdusResent	Todos estos atributos nos permiten conocer el estado actual de una conexión especifica que se encuentra operando en la capa de transporte que implementa la plataforma ISODE, y todos ellos están definidos acorde a la recomendación X.721 (1992) del CCITT.

Tabla # 22D Objeto transportConnection.

5.2.1.5 MIBiqa: la MIBiqa es la que implementa el "*proxy*" SNMP/CMIP que nos permite gestionar dispositivos SNMP a través del mapeo de peticiones CMIP dentro de peticiones SNMP, y traslada los "*traps*" y las respuestas SNMP dentro de eventos y respuestas CMIP. Las MIBs SNMPv1 que han sido trasladadas dentro de módulos SMI en OSIMIS utilizando la herramienta imibtool y que por tanto pueden ser gestionadas son mostradas en la Tabla # 23.

MIBs SNMP convertidas a módulos SMI
RFC1155-SMI
RFC-1212
RFC-1215
RFC1213-MIB
RFC1215-TRAP
RFC1354-MIB

Tabla #23 MIBs SNMP Trasladadas.

La Figura #43 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

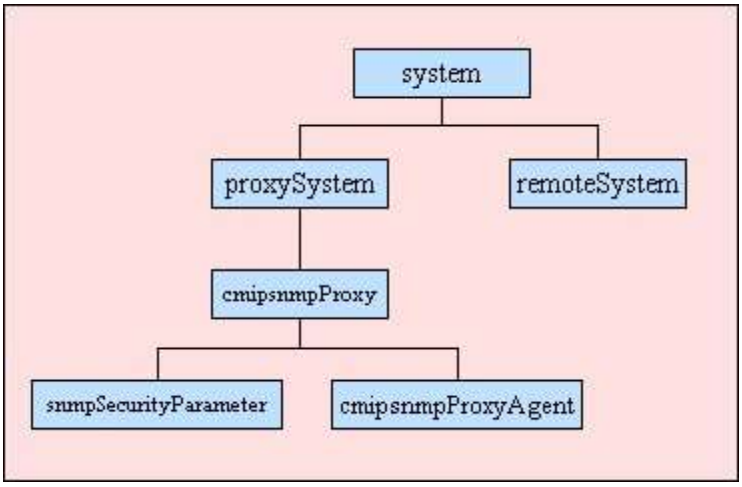


Figura # 43 Herencia jerárquica de los Objetos en la MIBiqa.

Objeto cmipsnmpProxyAgent:	
Este objeto nos permite representar cada objeto remoto a ser gestionado por medio de una instancia de este.	
Atributos	**Descripción**
1.cmipsnmpProxyAgentId 2. transportAddresses 3. managementProtocol 4. supportedMIBs 5. administrativeState 6.snmpMsgRetryLimit 7.snmpMsgTimeout	2. Almacena la dirección donde se manejan los mensajes SNMP, de la forma direcciónIP+Puerto. 3. Nos presenta el protocolo a ser gestionado, por ahora en nuestro caso solo puede SNMPv1. 4. Aquí se muestra la lista de MIBs SNMP que han sido instanciadas y sobre las cuales se puede realizar gestión. 6. Almacena el número de intentos para recuperar un mensaje. 7. Almacena el tiempo muerto que se espera un mensaje.

Tabla # 24A Objeto cmipsnmpProxyAgent.

Objeto cmipsnmpProxy:	
Este objeto lee los requerimientos de configuración inicial para poder crear un objeto cmipsnmpProxyAgent adecuado a esta configuración.	
Atributos	**Descripción**
1. cmipsnmpProxyId 2. snmpMsgRetryLimit 3. snmpMsgTimeout	2. Almacena el número de intentos para recuperar un mensaje. 3. Almacena el tiempo muerto que se espera un mensaje.

Tabla # 24B Objeto cmipsnmpProxy.

Objeto remoteSystem:	
Este objeto es el encargado de sondear la clase internetSystem del agente SNMPv1, y si este sondeo falla entonces no hay objeto remoto a representar y el objeto cmipsnmpProxyAgent es destruido, sin embargo si el sondeo es exitoso se crean los objetos que representen los MOs remotos.	
Atributos	**Descripción**
1. systemId 2. systemTitle 3. operationalState 4. usageState	

Tabla # 24C Objeto remoteSystem.

Objeto snmpSecurityParameter:	
Esta clase de objeto permite hacer uso de los parámetros de seguridad con que cuenta SNMP.	
Atributos	**Descripción**
1. snmpSecurityParameterId 2. snmpSecurity	2. Permite manipular los parámetros de seguridad del SNMP.

Tabla # 24D Objeto snmpSecurityParameter.

Objeto proxySystem:	
Esta clase es la encargada de representar los recursos locales del sistema.	
Atributos	**Descripción**
1. systemId 2. systemTitle 3. operationalState 4. usageState	

Tabla # 24E Objeto proxySystem.

5.2.1.6 MIBodp: La MIBodp define un objeto gestionado que permite demostrar como se puede usar el modelo de gestión OSI para efectuar llamadas a métodos remotos arbitrarios (acciones) sobre objetos de gestión distribuida.

La Figura #44 muestra la herencia jerárquica de los objetos definidos en esta MIB, y las tablas subsecuentes permiten conocer estos objetos.

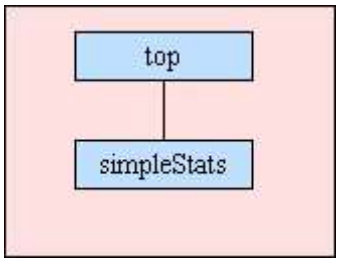


Figura # 43 Herencia jerárquica de los Objetos en la MIBodp.

Objeto simpleStats:	
Este objeto es quien provee servicios estadísticos simples en la forma de acciones. Es decir nos permite calcular algunas estadísticas a partir de los números dados usando la facilidad de ejecutar acciones de OSIMIS.	
Atributos	**Descripción**
1. simpleStatsId	
Acciones	**Descripción**
1. calcSqrt 2. calcMeanStdDev	1. Permite calcular la raíz cuadrada de un número real no negativo. 2. Permite calcular la media y desviación estándar de una serie de números reales.

Tabla # 25 Objeto simpleStats.

5.3 COMO CREAR OBJETOS GESTIONADOS Y AGENTES EN OSIMIS.

Esta sección es talvez de las más importantes de la monografía y nuestro mayor aporte en el desarrollo del trabajo de grado, puesto que aquí se conjugan la compilación e instalación de OSIMIS con el conocimiento de su infraestructura e implementación interna, para encausarlos en el desarrollo de nuestros propios agentes de gestión.

Antes de adentrarnos por completo en el tema, es importante en este momento resaltar que OSIMIS '**NO'** es una plataforma completa en el nivel de aplicación

(como ya lo habíamos expresado), más bien tiene la intención que los usuarios de esta ingresen y conozcan su código fuente para ayudar a desarrollarla y hacerla mas genérica en cuanto a los estándares se refiere. Como pudimos observar en el Capitulo III el paquete OSIMIS es bastante extenso, ocupando unos 100Mbytes de espacio en disco entre binarios y archivos fuente, por lo tanto nos podemos imaginar la cantidad de líneas de código en lenguaje C y C++ que posee, las cuales definen 254 clases y un número parecido de estructuras, lo cual hace el estudio de todo su código fuente una tarea más que difícil, por lo cual debemos concentrarnos solo en las de importancia fundamental.

Para lograr entonces nuestro objetivo de entender completamente el funcionamiento y estructura de OSIMIS, y poder utilizarlo como plataforma de gestión y de desarrollo de agentes de gestión y objetos gestionados debemos conocer la definición y funcionamiento de algunas clases primordiales (la mayoría ya fueron nombradas), y saber como podemos utilizar los métodos que estas definen; Las clases C++ que debemos conocer se clasifican en tres partes:

1. Clases para el soporte del proceso de coordinación, para escribir aplicaciones manejadas por eventos asíncronos:
   - Clase KS (Knowledge Source).
   - Clase Coordinator.
   - Clase ISODECoordinator.
2. Clases para el soporte del manejo transparente ASN.1:
   - Clase Attr.
   - Clase AnyType.
   - Clase AVA.
3. Clases para el soporte del sistema de gestión genérico (GMS):
   - Clase MOClassInfo.
   - Clase MO.
   - Clase Top.

Las clases para el soporte del proceso de coordinación son esenciales para aquellos gestores que reciben eventos asíncronos desde más de una fuente, por ejemplo de muchos agentes o un agente y una interfaz usuario etc. Las clases para el soporte ASN.1 permiten a las aplicaciones del programa de gestión intercambiar con el lenguaje de programación tipos de datos en lugar de constructores ASN.1 representando estructuras abstractas. Las clases para el GMS nos dan la infraestructura genérica necesaria para el funcionamiento de los objetos gestionados y la creación de nuevos.

Debido a que la definición de estas clases y sus métodos es extenso, estos se han trasladado al Anexo B que se puede encontrar al final de la monografía, recomendamos leer el anexo para comprender mejor este capitulo.

NOTA: Es fácil confundirse con clases MO y clases C++; recalcamos que las clases MO son representaciones de las clases C++, y las clases C++ se usan para muchas cosas más, como por ejemplo para definir los tipos de atributos.

5.3.1 CREACION DE OBJETOS GESTIONADOS.

Antes de embarcarnos en la discusión de “como implementar objetos gestionados”, debemos recordar que OSIMIS nos ofrece el sistema GMS y el soporte GDMO, los cuales trabajando juntos nos dan la base para lograr esta implementación. Miremos las características que no necesitamos implementar puesto que ya son proveídas por el GMS.

- Una implementación completa de CMIS/P (el MSAP) incluyendo el manejo apropiado de errores, soporte para respuestas enlazadas, etc.
- Un objeto agente de gran amplitud (CMISAgent) el cual maneja las peticiones CMIS entrantes, selecciona los MOs objetivos conforme a los nombres distinguidos, “*scoping*” y provee el filtrado de los constructores, y además aplica

las peticiones CMIS a los objetos seleccionados.

- ♣ Discriminadores de envío de eventos y objetos de control Log. (estos los conocemos como objetos eventForwardingDiscriminator y log).
- ♣ Una librería de clases C++ para tipos de atributos comunes como counter, gauge, etc. Así como codificadores y decodificadores para la sintaxis asociada con estos tipos. (estos son los tipos SMI definidos).
- ♣ Una clase C++ para un MO Genérico (clase MO). Esta clase incluye muchos métodos importantes concernientes a la selección de objetos y acceso a los atributos.
- ♣ Clases C++ para clases MO "Estándar" tales como Top y System. (estas son las clases de objetos top y system).
- ♣ Asistencia para ordenar cronológicamente la comunicación entre instancias MO y los "recursos reales" que estos representan. (esto lo hace el Coordinador).

El siguiente ejemplo puede ayudarnos a clarificar el papel del GMS, Supongamos que una petición M-GET de CMIS es recibida por el agente:

1 La petición es analizada gramaticalmente y la sintaxis es chequeada.
2. Como resultado de la aplicación de reglas para nombrado, "*scoping*" y filtrado, un conjunto de instancias de MO objetivo son construidas.
3. Para cada una de estas instancias MO el método genérico GET es llamado con un parámetro el cual especifica una lista de OID's para los atributos objetivo.
4. Para cada uno de los atributos objetivo el método GET apropiado es llamado y el correspondiente valor del atributo es retornado.
5. Finalmente, el agente usa los valores recuperados para aumentar y codificar el mensaje M-GET respuesta.

Para una implementación de MO típica todo el código que se invocará en este ejemplo es proporcionado con el GMS, así el usuario GMS es liberado de toda la

responsabilidad por operaciones concernientes con el propio agente y con aquellos que existan en la frontera del agente o MO.

Recordemos que un sistema de gestión basado en GMS funciona como un simple proceso UNÍX. Resumiendo lo estudiado en Capítulos anteriores el GMS tiene los siguientes tipos principales de componentes:

- Agente CMIS (CMISAgent): Este es un objeto C++ el cual personifica el papel del agente como esta definido en el estándar de gestión OSI. Siempre hay solo una instancia de este objeto presente en un sistema.

- Coordinador (Coordinator): Este es un objeto C++ el cual maneja la comunicación y las señales dentro del sistema. Siempre hay una instancia sólamente de este objeto es un sistema.

- Objetos Gestionados (MOs): Estos son objetos C++ que envuelven la información de gestión y el comportamiento especificado en la definición de la MIB pertinente. Generalmente hay un objeto C++ instanciado por cada MO instanciado en la jerarquía de contenimiento.

- Fuente de Conocimiento (KS): Estos son objetos C++ que tienen que ver con la interconexión de los objetos gestionados al coordinador.

Como sabemos el agente CMIS y el coordinador son suministrados como componentes completos del GMS, mientras las fuentes de conocimiento (KS) y los objetos gestionados (MO) deben ser derivados específicamente para la MIB que será implementada desde las versiones genéricas proveídas por el GMS.

Los pasos a seguir para la creación de nuevos objetos gestionados son mostrados en la Figura #44, como ya estudiamos la manera en que trabaja el compilador GDMO nos olvidamos de los "*scripts*" y bases de datos que no debemos manipular y que son utilizados por este.

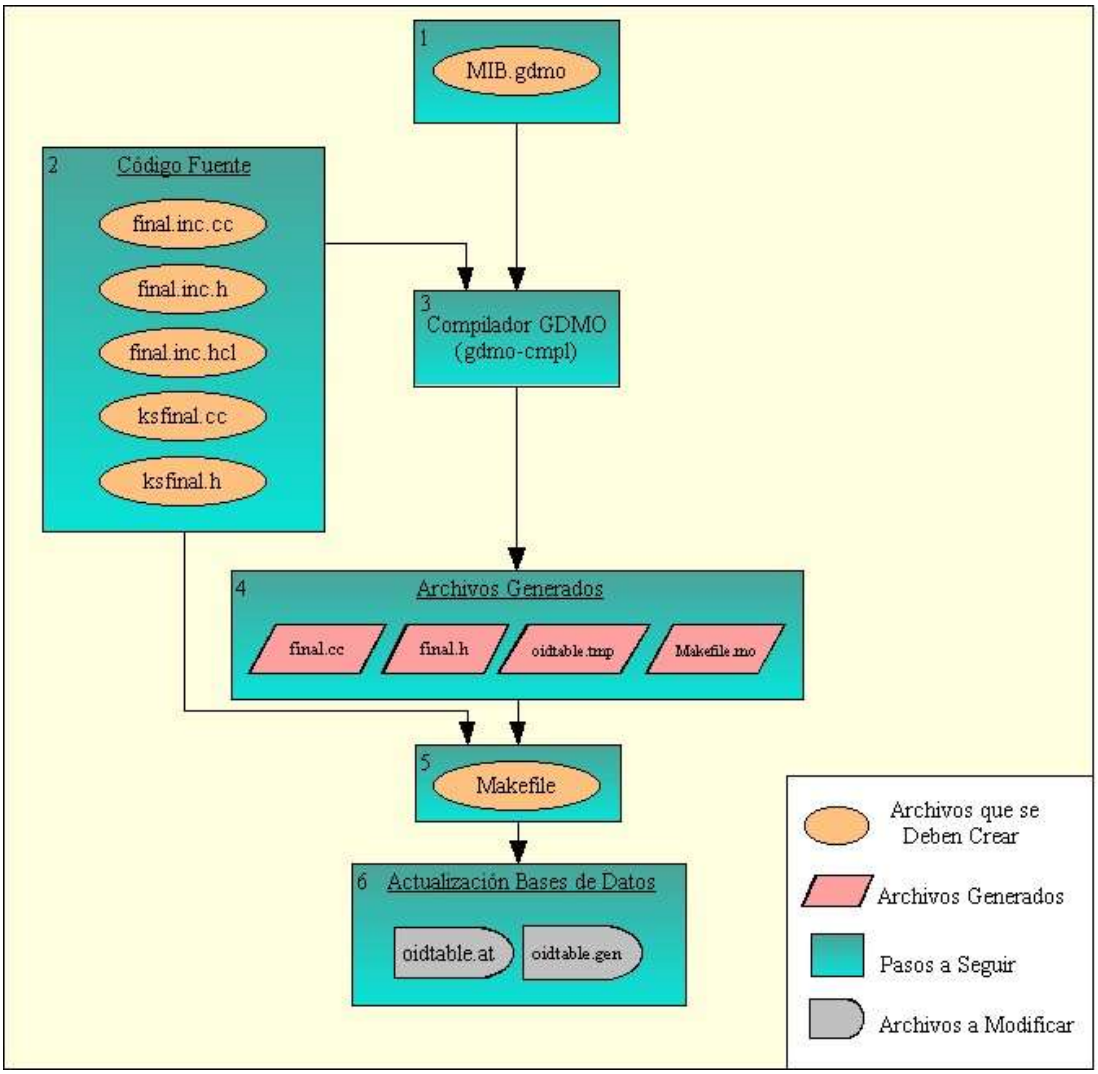


Figura #44 Pasos Para la Creación de Objetos Gestionados.

1. Debemos crear una MIB en formato GDMO para definir la nueva clase o clases, la cual incluirá la identidad de la clase MO padre en la herencia jerárquica, una lista de los atributos adicionales que la nueva clase contiene, etc.
2. Crear el código fuente necesario para adicionar el comportamiento deseado al MO, esto es, crear los tres archivos que espera el compilador GDMO

para ser combinados con el código generado por este, como vimos antes estos concatenan el nombre del MO con las extensiones “.inc.cc”, “.inc.h” e “.inc.hcl”; además como nuestro objeto gestionado requiere de un KS especifico también debemos crear el código para este aquí.

3. Utilizar el compilador GDMO para compilar (1).
4. Los archivos <nombre del MO>.cc , <nombre del MO>.h, Makefile.mo, y oidtable.tmp, que son generados por (3).
5. Crear el archivo Makefile especifico para generar la librería, y compilar con este (4) y (2).
6. Actualizar las bases de datos de OSIMIS.

Estos seis puntos anteriores serán desglosados a continuación, basándonos en un objeto agente creado por nosotros que llamamos “final”, se sugiere que las siguientes secciones sean leídas junto con el Anexo D donde se encuentran todos los archivos que se crearon para la implementación de tal MO.

5.3.1.1 Creación de la MIB: Este es el punto de partida para definir nuestro objeto y poder darle luego el comportamiento deseado, por lo tanto antes de crear la MIB, debemos tener muy claro que queremos de nuestro objeto y como queremos interactuar con el.

Vamos a crear entonces un objeto bastante completo que llamaremos “final”, que implemente las tres formas de comunicación con el recurso real, que permita la realización de acciones, y que realice el reporte de eventos y notificaciones. Nuestro punto de inicio para lograr esto es definir la MIB con los atributos, notificaciones, acciones etc, que sean necesarios; en el Anexo D podemos observar la definición de la MIB, la cual es llamada MIBfinal, y a continuación como en el caso de las MIBs que ya trae OSIMIS integradas, presentaremos una explicación de esta en la Figura #45 y la Tabla #26 asumiendo desde ya el comportamiento que luego le será asignado al objeto, esto con el fin de lograr una mejor explicación.

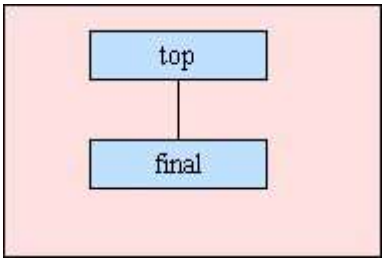


Figura # 45 Herencia jerárquica de los Objetos en la MIBfinal.

Objeto final:	
El objeto "final" nos permite gestionar el estado actual de la unidad de CD, cuenta con una acción que nos permite montarla y desmontarla a nuestro gusto, realiza un "*polling*" sobre el estado de la unidad de CD, recibe eventos de una aplicación de usuario externa que queremos gestionar, en este caso un simple número entero sobre el cual tiene asociados un umbral y TideMark (limite extremo) para generar notificaciones, permite el reporte de eventos, y también acepta la comunicación con el recurso real bajo una petición externa.	
Atributos	**Descripción**
1. finalId 2. cadenamodificable 3. estadocdrom 4. variableusuario 5. usuarioumbral 6. usuariomarca	1. El identificador de una instancia de la clase. 2. Este es un atributo del tipo GraphicString que usamos para ejemplificar la posibilidad de obtener, fijar y reemplazar un atributo. El valor de este atributo es una cadena arbitraria. 3. Este es un atributo del tipo GraphicString y lo usamos para conocer el estado de la unidad de CD, asociándole un mecanismo "polling". 4. Este atributo del tipo Gauge almacena el valor de una variable que reportara una aplicación de usuario mediante un evento. 5. Atributo del tipo Gauge-Threshold asociado al atributo variableusuario, con lo cual podemos asociar umbrales a esta variable. 6.atributo del tipo TideMark asociado al atributo variable usuario, lo cual nos permite adicionar el manejo del limite extremo.
Acciones	**Descripción**
1. montarcd	1. Esta acción nos permitirá realizar el montaje y desmontaje de la unidad de CD.
Notificaciones	
1. objectCreation	

2. objectDeletion
3. attributeValueChange
4. qualityofServiceAlarm

Tabla # 26 Objeto final.

5.3.1.2 Creación del código fuente: Como ya observamos, deseamos cierto comportamiento especifico para nuestro objeto, por lo cual debemos incluir código escrito por nosotros, esto es, debemos crear los archivos ".inc.cc", ".inc.h" y ".inc.hcl" necesarios para adicionar el comportamiento, además como nuestro objeto tendrá una comunicación de "activación por eventos" con el recurso real, es necesario crear una fuente de conocimiento especifica (KS) que le permita al coordinador de OSIMIS manejar estos eventos.

Empecemos entonces por estudiar como OSIMIS nos facilita la manera en que creamos estos archivos a través del uso de los métodos polimorfos que tiene definidos, los cuales podemos observar en el Anexo B, redefiniendo estos métodos polimorfos en las clases de objetos derivadas es que logramos ese comportamiento especifico que queremos.

A continuación observaremos el conjunto de métodos polimorfos que pueden ser redefinidos por clases derivadas, todos estos métodos retornan valores del tipo "int", y solo pueden tomar los valores OK en un éxito y NOTOK en una falla (0 y -1 respectivamente) que ya han sido definidos dentro del código de OSIMIS usando "#define".

virtual int get (int attrId, int classLevel, AVA*& errorInfo, Bool checkOnly, int operId);

El método get debe ser redefinido solamente cuando los atributos de la clase necesitan ser actualizados antes de responder a una petición M-GET de CMIS. Esto es necesario cuando un régimen de "acceso bajo una petición externa" es

ejercido respecto al recurso real asociado. El método debe recuperar la información necesaria y actualizar la petición de atributos, usando los métodos de la clase Attr (setval, replval) o haciendo a la medida métodos de sus clases derivadas.

a*ttrId:* Es el índice integer de ese atributo en el array de atributos para la clase.

classLevel: Es el nivel de la clase en el árbol de herencia jerárquica (0 para Top).

checkOnly: **Es un argumento adicional, que se usa como bandera booleana para permitir al GMS el ejercer atomicidad.**

operId: Es un argumento adicional, que se usa como bandera de tipo integer para habilitar al GMS ha manejar respuestas asíncronas.

virtual int set (CMISModifyOp setMode, int attrId, int classLevel, void* newAttrValue, AVA*& errorInfo,Bool checkOnly, int operId);

El método set debe ser redefinido solo si una operación set sobre un atributo de gestión debe resultar en modificación de la información en el recurso real asociado. Es de notar que pueden haber atributos para los cuales una operación set pueda no disparar una actualización o acción asociada al recurso real, tal como umbrales etc. En este caso se requiere que sean manejadas por las partes genéricas de la clase MO.

setMode: Es el modo de operación set, estos modos pueden ser pueden ser m_replace, m_addValue, m_removeValue o m_setToDefault.

attrId: Es el índice integer de ese atributo en el array de atributos para la clase.

classLevel: Es el nivel de la clase en el árbol de herencia jerárquica (0 para Top).

newAttrValue: Es el valor a ser usado en la operación set, los mismos valores serán usados por el GMS para fijar el atributo actual si un error es retornado, por lo tanto este NO debe ser liberado.

virtual int action (int actionId, int classLevel, void* actionInfo, void*& actionReply, Bool& freeFlag, AVA*& errorInfo, Bool checkOnly, int operId);

Este método es similar a get y set respecto a los argumentos *actionId* y *classLevel* y el acceso al recurso real, la única diferencia con set es que en action hay una sola acción contrario a los muchos atributos que set posiblemente necesite para una operación y una respuesta.

action*Id:* Es el índice integer de ese atributo en el array de atributos para la clase.

classLevel: Es el nivel de la clase en el árbol de herencia jerárquica (0 para Top).

actionInfo: Es la información de la acción a ser desempeñada.

actionReply: Es la respuesta que debe se enviada (si hay alguna). No se debe liberar el actionInfo y se debe asegurar que actionReply tenga memoria asignada.

freeFlag: Es un argumento del tipo booleano adicional para dar más flexibilidad a los métodos pero no adiciona nueva funcionalidad.

checkOnly: **Es un argumento adicional, que se usa como bandera booleana para permitir al GMS el ejercer atomicidad.**

operId: Es un argumento adicional, que se usa como bandera de tipo integer para habilitar al GMS ha manejar respuestas asíncronas.

virtual int deleteRR (AVA*& errorInfo, void* deleteInfo, Bool checkOnly, int operId);

Este borrará el objeto gestionado si puede ser borrado a través de CMIS, según la especificación de la clase, esto puede ser permitido solo si no contiene objetos. En este caso, los objetos contenidos deben ser obtenidos usando el método getSubordinates de la clase MO (como se puede ver en el Anexo B) con “True” como argumento para posiblemente comprobar el recurso real. Si se conoce que la clase tiene objetos subordinados y estos deben ser borrados, hay que prestar atención en el constructor. Una implementación típica de este método chequea por posibles objetos subordinados que pueden resultar en un retorno de error al intentar ser borrados. El destructor debe tomar cuidado de realizar cualquier limpieza, como interacciones con el recurso real y liberación de memoria.

deleteInfo: Contiene la información necesaria sobre la instancia de objeto a ser borrado.

checkOnly: **Es un argumento adicional, que se usa como bandera booleana para permitir al GMS el ejercer atomicidad.**

operId: Es un argumento adicional, que se usa como bandera de tipo integer para habilitar al GMS ha manejar respuestas asíncronas.

virtual int buildReport (int eventId, int classLevel, void*& eventInfo, Bool& freeFlag);

Este es usado para construir un reporte de eventos a fin de ser enviado como un log de OSIMIS, este método fue creado como polimorfo ya que el compilador GDMO no soporta la definición de otro tipo de notificaciones además de las ya definidas (objectCreation, objectDeletion, etc.), entonces con buildReport nosotros podemos crear otro tipo de notificaciones, como por ejemplo el desborde de un umbral, el traspaso de un número determinado de cambios en un atributo etc.

eventId: Es el índice integer del evento a ser informado.

classLevel: Es el nivel de la clase en el árbol de herencia jerárquica (0 para Top).

eventInfo: Es un apuntador a una estructura que contendrá la información del evento.

freeFlag: Es un argumento del tipo booleano adicional para dar más flexibilidad a los métodos pero no adiciona nueva funcionalidad.

virtual int refreshSubordinates (AVA*& errorInfo,int operId);

La existencia de algún objeto gestionado es solo conocida accediendo el recurso real asociado. Esto es verdadero por ejemplo para recursos que no pueden notificar al sistema gestor cuando los objetos son creados o borrados, como por ejemplo para administrar entradas de tablas en un Kernel del sistema operativo o en un recurso suavemente acoplado, en este caso, la respuesta a peticiones CMIS debe ser echa en dos vías, usando un esquema "*polling*" o simplemente un régimen de "acceso bajo una petición externa".

El esquema "*polling*" tiene el inconveniente de introducir tráfico entre el sistema gestionado y el recurso real. El esquema de "acceso bajo una petición externa"

rectifica este problema pero no puede soportar notificaciones e incrementa el tiempo de respuesta para peticiones CMIS.
operId: Es un argumento adicional, que se usa como bandera de tipo integer para habilitar al GMS ha manejar respuestas asíncronas.

virtual int refreshSubordinate (RDN rdn, AVA*& errorInfo,int operId);
Este es exactamente el mismo que el anterior pero para un subordinado particular el cual es identificado a través de su nombre distinguido relativo. En este caso, solo el objeto especificado debe ser refrescado el cual puede crearlo, borrarlo o ninguno de los dos. En el último caso, es mejor refrescar sus datos (atributos) a fin de evitar después otro acceso al recurso real.
rdn: Contiene el nombre distinguido relativo del objeto subordinado.
operId: Es un argumento adicional, que se usa como bandera de tipo integer para habilitar al GMS ha manejar respuestas asíncronas.
Como acabamos de observar existe en estos métodos un parámetro especial llamado errorInfo el cual contiene:

1. Un código CMISError.
2. Un identificador, es decir un nombre como está registrado en oidtable.at.
3. Un valor arbitrario, es decir un apuntador a la clase Attr* con un valor correspondiente al identificador.

El código CMISError siempre debe estar presente cuando un NOTOK es retornado para decirle al GMS cual fue el error, y los valores permitidos para este son mostrados en la Tabla #27.

Método	Valores de CMISError
createRR	invalidAttributeValue, missingAttributeValue, processingFailure.
deleteRR	ProcessingFailure.
get	processingFailure.

set	invalidAttributeValue, invalidOperation, processingFailure.
action	invalidArgumentValue, processingFailure.
refreshSubordinate(s)	processingFailure.

Tabla # 27 Valores de Error Para los Métodos Polimorfos.

Con processingFailure uno puede pasar un identificador y/o un valor arbitrario, para informar a la aplicación remota de que salió mal exactamente. Si processingFailure no es proveído, el GMS lo llenará con valores generales (los cuales no dicen mucho) como es ordenado por CMIS.

El identificador debe ser proveído en el caso de un error invalidAttributeValue o missingAttributeValue en createRR. Notemos que los métodos asignan el espacio para la información del error el cual es liberado por el GMS.

También es bueno aclarar que nos permiten hacer exactamente los parámetros checkOnly y operId, los cuales han sido creados pensando en futuras versiones.

La bandera booleana checkOnly es el parámetro usado para atomicidad, pero por el momento este debe ser mas o menos ignorado ya que el GMS aun no soporta atomicidad. Pero es muy recomendable para escribir un buen código adicionar al comienzo del método respectivo (después de llamar al mismo método padre), un camino de decisión como el mostrado en la Figura #46, de esta forma se rechazaran las peticiones atómicas para esta clase, y podrán ser adicionadas en un futuro.

```
if (checkOnly)
    return NOTOK;   // no es necesario fijar el error en este caso
```

Figura #46 Recomendación Para Usar checkOnly.

El parámetro operId puede ser ignorado para todas las clases que están usando la interfaz sincrónica GMS, es decir todo, puesto que este parámetro será usado por clases que usen la API asíncrona que se piensa desarrollar en un futuro para OSIMIS.

5.3.1.2.1 Creación del archivo "final.inc.cc": El archivo final.inc.cc se convierte en la base para definir el comportamiento del objeto, y es en este que debemos redefinir los métodos polimorfos que necesitemos para que nuestro objeto actué y se comunique con el recurso real como lo deseamos. Ahora miremos de manera global en la Tabla # 28 cuales métodos debemos redefinir, el porque, y que pasos debemos seguir para hacerlo.

Método Polimorfo a Redefinir	Razón Para Hacerlo y Pasos a Seguir
createRR	Este método siempre debe ser redefinido para todo objeto que sea creado, puesto que nos permite realizar todas la inicializaciones necesarias del propio objeto, de sus atributos etc, por lo tanto para nuestro objeto "final" también debe ser redefinido y en este debemos realizar los siguientes pasos: • Inicializar la clase superior si aun no lo esta. • Inicializar los diferentes atributos que lo necesitan como por ejemplo alarmas, variables etc, y para esto debemos conocer el tipo exacto. • Inicializar la instancia de la clase. • Debemos realizar la asociación de los atributos con sus umbrales respectivos. • Como nuestro objeto necesita de una fuente de conocimiento para la comunicación con recursos reales, esta debe ser inicializada. • La fuente de conocimiento debemos también registrarla. • Debemos realizar la programación de los wake-ups que manejaran el coordinador si lo necesitamos.
deleteRR	Necesitamos redefinir este método para poder destruir nuestra instancia de objeto asegurando el mantener una herencia jerárquica adecuada. • Debemos obtener el nombre exacto de la instancia.

	• Debemos liberar la memoria que ocupan las estructuras que se usan en el objeto. • La fuente de conocimiento debe ser desregistrada. • Por último pasamos a eliminar la instancia de objeto.
get	Como nuestro objeto va a permitir la comunicación con el recurso real de acceso bajo una petición externa, debemos redefinir este método también, y las tareas que debemos hacer allí son: • Hay que chequear si la petición get se efectúa sobre un atributo definido en este objeto o en su superior, si pertenece al superior debemos llamar su método correspondiente. • Debemos detectar exactamente cual es el atributo y realizar la operación sobre este.
set	La redefinición de este método es necesario ya que nuestro objeto posee atributos que pueden ser fijados. • Hay que chequear si la petición set se efectúa sobre un atributo definido en este objeto o en su superior, si pertenece al superior debemos llamar su método correspondiente. • Debemos detectar exactamente cual es el atributo y realizar la operación sobre este.
action	Nuestro objeto tiene definida una acción por lo tanto debemos también redefinir este método. • Hay que chequear si la petición de realizar una acción se efectúa sobre este objeto o sobre una acción de su superior, si pertenece al superior debemos llamar su método correspondiente. • Debemos detectar la acción y realizar la operación correspondiente.
buildReport	Nuestro objeto tiene definidos un umbral y tide-mark sobre un atributo, la redefinición de este método nos permite reportar los eventos. • Debemos asegurarnos si la construcción del reporte puede ser hecha aquí o si se debe llamar al método de la clase superior. • Detectamos que evento ocurrió y enviamos la información necesaria, como por ejemplo que atributo generó el reporte, el valor de este, el

	objeto al que pertenece.

Tabla # 28 Métodos Polimorfos a Redefinir en el Código Fuente.

Como no hemos definido objetos subordinados para nuestro objeto, no tenemos necesidad de redefinir los métodos refreshSubordinate(s), pero si debemos redefinir otros métodos particulares dentro de final.inc.cc para complementar y efectuar correctamente las acciones que deseamos. En el Anexo D podemos observar el código fuente de final.inc.cc que desarrollamos, el cual se encuentra muy bien comentado y sigue los pasos plasmados en la Tabla #27, el cual junto con los objetos que ya define OSIMIS podremos usar como ejemplo para la creación del código fuente de nuevos objetos gestionados.

5.3.1.2.2 Creación del archivo "final.inc.h": Dentro de este archivo debemos definir los métodos y valores constantes que se utilizaran en la clase, lógicamente debemos ser muy cuidadosos al definir los métodos colocándolos dentro de la etiqueta correspondiente, public, protected o private, y para hacer mas ordenado, legible y expansible el código debemos declarar aquí los valores constantes que utilicemos en final.inc.cc; en el Anexo D podemos observar de manera muy explicita el archivo final.inc.h que creamos específicamente para definir nuestro objeto.

5.3.1.2.3 Creación del archivo "final.inc.hcl": Este archivo como ya explicamos es usado para definir de que clases es heredada la nuestra, por lo tanto su creación es muy sencilla y consta de un par de líneas para definir la herencia e incluir los archivos cabecera necesarios; en el Anexo D podemos observar el archivo final.inc.hcl especifico para nuestro objeto.

5.3.1.2.4 Creación de la fuente de conocimiento: Recordemos que nuestro objeto permite los tres métodos de acceso al recurso real, y ya vimos como el

método de “acceso bajo una petición externa” fue proporcionado con la redefinición del método get en el archivo final.inc.cc, ahora necesitamos como ya lo habíamos notado de una fuente de conocimiento derivada específicamente para nuestro objeto, que nos permita la implementación de los otros dos métodos de comunicación con el recurso real. Como ya lo hemos aclarado, todas las implementaciones de MO deben hacer uso de las facilidades proveídas por el coordinador para el manejo de sus eventos y comunicaciones (de otra manera la operación correcta no puede ser garantizada), y las principales facilidades ofrecidas por el coordinador para realizar estos métodos son:

- ➔ El llamado de una rutina proporcionada al usuario GMS a intervalos regulares (usada para soportar un régimen “*polling*”).
- ➔ El monitoreo de un descriptor de archivos y llamado de una rutina proporcionada al usuario GMS cuando ocurre un evento de comunicación (usado para soportar un régimen de activación por eventos).

Y la manera de utilizar esta facilidades es a través de la derivación desde la clase de objeto fuente de conocimiento genérica que se encuentra definida en “/osimis/kernel/GenericKS.h.” y que podemos observar en el Anexo B dentro de la clase KS, esta clase provee algunos métodos estándar que pueden ser invocados por el coordinador cuando algún evento significante ocurre, y los que necesitaremos utilizar nosotros son los dos siguientes:

1. KS:readCommEndpoint (int): El cual es invocado por el coordinador cuando los datos están disponibles para ser leídos. Este es el mecanismo usado por un régimen de activación por eventos para recibir mensajes desde el recurso real.

2. KS: wakeUp (char*): El cual es invocado por el coordinar cuando un evento timer ocurre. Este es el mecanismo usado por un régimen manejado por “*polling*”.

Por consiguiente debemos redefinir estos dos métodos de la clase KS en nuestra fuente de conocimiento especifica, la cual consta de los dos archivos que veremos a continuación.

5.3.1.2.4.1 Creación del archivo "ksfinal.cc": En este archivo se define la fuente de conocimiento especifica, la cual es derivada desde la fuente de conocimiento genérica proveída por OSIMIS, aquí debemos definir el comportamiento o acciones que se realizaran cuando un evento timer ocurre, en nuestro caso este ocurrirá según lo programado en final.inc.cc, así como también cuando hay información entregada desde la aplicación usuario lista para ser leída, por lo tanto debemos redefinir los métodos readCommEndpoint y wakeUp de la clase KS genérica, y con ellos debemos crear otros necesarios para asegurar su correcto funcionamiento, es decir que se encarguen de tareas adicionales como alistar el puerto necesario para la escucha, registrar y desregistrar la instancia de objeto que utilizara la fuente de conocimiento. La implementación de este archivo la podemos estudiar en el Anexo D.

5.3.1.2.4.2 Creación del archivo "ksfinal.h": Este es el archivo cabecera necesario para la definición de la fuente de conocimiento, la cual hemos llamado ksfinal, y en este debemos definir la clase, su herencia, los archivos cabecera necesarios y los métodos que usaremos colocando mucho cuidado con las etiquetas public, private y protected, además también definimos aquí ciertas variables que usamos para hacer mas legible y extensible el código como es recomendado, la implementación exacta de este archivo para nuestro caso puede ser estudiada en el Anexo D.

5.3.1.3 Compilación con GDMO.

En este punto ya tenemos creados los archivos que el compilador GDMO busca en el momento de realizar la compilación, por lo tanto nuestro paso siguiente es

bastante sencillo y como se explico en la sección del compilador GDMO consiste de ejecutar la siguiente línea estando dentro del directorio en el cual se crearon todos los archivos necesarios para el objeto, que en nuestro caso es "/tesis/osimis/agent/final".

$ gdmo-cmpl MIBfinal

5.3.1.4 Archivos generados por el compilador GDMO.

Como ya lo habíamos expresado el uso del compilador nos generara los archivos final.cc, final.h, oidtable.tmp y Makefile.mo que pueden ser observados también en el Anexo D. Es muy productivo para nosotros en este momento observar en el archivo final.h el corazón de la definición de la clase que se hace en las líneas mostradas en la Figura #47.

```
...⇒⇒⇒⇒⇒ Línea #47
    static const int
    I_finalId,
    I_cadenamodificable,
    I_estadocdrom,
    I_variableusuario,
    I_usuarioumbral,
    I_usuariomarca,
#define I_finalNATTRS    6
    I_objectCreation,
    I_objectDeletion,
    I_attributeValueChange,
    I_qualityofServiceAlarm,
#define I_finalNNOTIFS    4
    I_montarcd;
#define I_finalNACTIONS    1

    Attr        *_attrs[I_finalNATTRS];
```

Figura #47 Definición de la clase "final" en el Archivo final.h

En estas líneas se asocian los 6 atributos de la clase final con un conjunto de enteros y se declara un arreglo de punteros a los mismos atributos. Un punto clave es que los 6 atributos, las 4 notificaciones y la acción declarada aquí son solamente aquellos que la clase final posee “adicionalmente” a aquellos heredados desde su superior (Top).

Además como ya se ha visto, a fin de que la clase C++ refleje adecuadamente el comportamiento de las correspondientes clases MO, una cierta cantidad de información tiene que estar disponible en tiempo de ejecución la cual normalmente esta disponible solo en el momento de la compilación. Por ejemplo: Una instancia MO necesita conocer no solamente cual es su clase padre sino también cuales son las clases de este ancestro, esta debe tener también acceso al mapeo entre los enteros identificadores de atributos abordados antes y los OID de atributos desde la definición de la MIB. Por tal razón antes que duplicar esta información en cada instancia de la clase MO, una simple instancia de la clase MOClassInfo es creada por cada clase MO instanciada dentro del sistema, por lo tanto podemos observar en la Figura #48 como al comienzo de la definición de la clase final dentro del archivo final.h, una instancia estática de la clase MOClassInfo es declarada:

```
...⇒⇒⇒⇒⇒ Línea #39
   static MOClassInfo* _classInfo;
...
```

Figura #48 Declaración de la instancia MOClassInfo en el Archivo final.h

Las anteriores son las definiciones más relevantes en cuanto a los archivos final.cc y final.h, los archivos Makefile.mo y oidtable.tmp los analizaremos mas adelante.

5.3.1.5 Creación del archivo Makefile.

La creación del archivo Makefile es necesaria para ser consecuentes con la manera en que se efectúa la compilación del código fuente en OSIMIS, y de esta manera hacer más flexible la integración con los "*scripts*" que se utilizan en OSIMIS, nos referimos exáctamente al CONFIG.make, el cual como sabemos define los caminos a librerías, archivos cabecera, opciones para los compiladores, tal como lo vimos en el Capitulo III. La creación del archivo Makefile debe ir acompañada de otro archivo llamado make, el archivo make es muy sencillo y sigue el formato de todos los que se usan en OSIMIS, en el solo debemos asegurar la ejecución correcta de la utilidad make de Linux, y a continuación la ejecución del CONFIG.make y nuestro Makefile, el archivo make creado se encuentra en el Anexo D.

Ahora para la creación de nuestro Makefile podemos tomar como ejemplo cualquiera de los Makefile que utilizan los MO que trae por defecto OSIMIS y el Makefile.mo[3] que genera el compilador GDMO, y así definir en nuestro "*script*" las opciones de limpieza de archivos, de instalación, de compilación, y variables de entorno necesarias; el Makefile creado se puede observar en el Anexo D y se encuentra muy bien comentado para su mejor entendimiento.

Ya en este punto hemos efectuado la creación de todos los archivos necesarios para formar nuestro objeto y para lograr integrarlo en OSIMIS, ahora debemos generar las librerías y posicionar estas y los archivos cabecera en la ubicación adecuada dentro de OSIMIS, de estos pasos se encarga nuestro Makefile ya creado, y la forma en que lo debemos usar es la siguiente:

---

[3] **El Makefile.mo no puede ser utilizado como el Makefile de compilación puesto que allí no se incluye la compilación de nuestra fuente de conocimiento, ni se incluyen las etapas de instalación, pero puede ser tomado como un buen ejemplo.**

5.3.1.5.1 Compilación: Estando dentro de la carpeta que contenga los archivos del objeto (para nuestro caso “tesis/osimis/agent/final”) escribimos la siguiente línea:

“./make all“

lo cual como ya aclaramos correrá el make que creamos y ejecutara adecuadamente nuestro Makefile, esto generara una librería conteniendo el MO que como podemos ver dentro del código del Makefile se llamara “libfinalmib.a“.

5.3.1.5.2 Instalación: Ahora debemos ubicar la librería y los archivos cabecera en su posición adecuada, lo cual hacemos de la siguiente forma:

“./make install-lib”	(instala la librería en su ubicación correcta).
“./make install-h”	(instala los archivos cabecera en la ubicación correcta).

Como observamos los pasos se hacen muy sencillos luego de la creación correcta de nuestro propio “*script*” Makefile.

5.3.1.6 Actualización de las bases de datos.

Como recordaremos OSIMIS posee unas bases de datos que utiliza comúnmente, llamadas oidtable.at y oidtable.gen que se encuentran ubicadas luego de nuestra correcta instalación de OSIMIS en la ruta “/osimis/etc”, estas bases de datos deben ser actualizadas con la información correspondiente a los nuevos atributos, nombres enlazados etc, de nuestra nueva clase de objeto; esta actualización la hacemos usando la información que encontramos en el oidtable.tmp generado por el compilador GDMO, en oidtable.tmp que puede ser observado en el Anexo D, esta claramente separada la información para cada oidtable, lo único que debemos hacer es copiar y pegar esta información en la respectiva base de datos.

Cumpliendo con todos los anteriores pasos ya tenemos lista una nueva clase de objeto gestionado, la cual para poder ser usada solo debe ser implementada dentro de un agente de gestión.

5.3.2 CREACION DE AGENTES.

Para la creación de Agentes de gestión, OSIMIS nos proporciona el código del agente SMA, el cual debe ser tomado no como ejemplo, sino como la base para la realización de cualquier otro agente, este código está representado por los archivos sma.h y sma.cc que se encuentran en la ruta "/tesis/osimis/agent/sma" y que han sido transcritos en el Anexo D para permitir su estudio y realizar comparaciones acerca de los pocos cambios necesarios en estos para generar el nuevo agente. A continuación en la Figura #49 veremos los pasos necesarios para crear el nuevo agente.

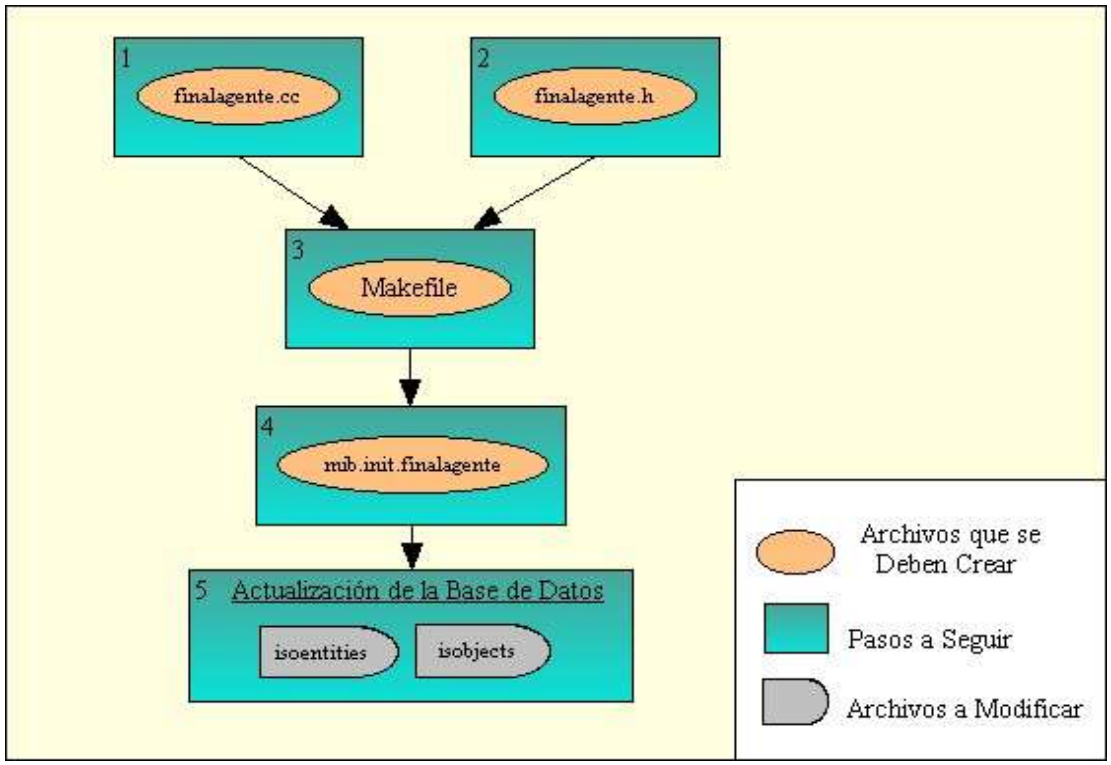


Figura #49 Pasos Necesarios Para Crear un Agente.

1. Crear el archivo fuente de nuestro agente.
2. Crear el archivo cabecera de nuestro agente.
3. Crear el "*script*" Makefile correspondiente y con este compilar e instalar el agente.
4. Crear el archivo que contenga la inicialización de la información de objetos definidos en la MIB o MIBs.
5. Actualizar las bases de datos.

5.3.2.1 Creación del archivo fuente: Para la creación de este archivo como ya aclaramos debemos usar como base el archivo sma.cc, en el cual únicamente debemos cambiar la línea mostrada en la Figura #50

```
...⇒⇒⇒⇒⇒ Línea #23
#include <stdlib.h>
#include "msap.h"
#include "Error.h"
#include "Coordinator.h"
#include "CmisAgent.h"
#include "Sma.h"              → DEBEMOS BORRAR ESTA LINEA
#include "finalagente.h"      → DEBEMOS ADICIONAR ESTA LINEA
...
```

Figura #50 Creación del Archivo finalagente.cc

Luego debemos renombrar este archivo con el nombre de nuestro agente, para nuestro caso finalagente.cc, esto se puede ver más claramente en el Anexo D.

5.3.2.2 Creación del archivo cabecera: Para la creación de este archivo tomamos como base el archivo sma.h y debemos realizar ciertos cambios significativos, además de renombrarlo con el nombre del archivo cabecera que incluimos anteriormente en los cambios a sma.cc, esto es, con el nombre finalagente.h, los cambios que se deben efectuar son mostrados en las Figuras #51A, #51B y #51C , estos incluyen adicionar el archivo cabecera de nuestra clase de objeto gestionado (Figura #51A), modificar el nombre de la aplicación, es decir cambiar el nombre del agente SMA por el de nuestro agente (Figura #51B),

modificar la línea donde se señala la tabla que contiene la información de inicialización de la MIB correspondiente al objeto (Figura #51B), y por último incluir los métodos de obtención de información de la clase e inicialización de la clase, para que el agente se encargue de crear el árbol de herencia (Figura #51C)

```
...⇒⇒⇒⇒⇒ Línea #24
#include "Create.h"
#include "SmiSntx.h"

#include "System.h"
#include "final.h"          → DEBEMOS ADICIONAR ESTA LINEA
```

Figura #51A Adición del Archivo Cabecera.

```
...⇒⇒⇒⇒⇒ Línea # 58
char*     service = "SMA";                  → DEBEMOS BORRAR ESTA LINEA
char*     service = "finalagente";          → DEBEMOS ADICIONAR ESTA LINEA

#define   MIB_INIT_FILE   "mib.init.sma"          → DEBEMOS BORRAR ESTA LINEA
#define   MIB_INIT_FILE   "mib.init.finalagente"  → DEBEMOS ADICIONAR ESTA LINEA
```

Figura #51B Inclusión del Nombre de la Aplicación e Inicialización de la MIB.

```
...⇒⇒⇒⇒⇒ Línea # 67
MOClass moclasses[] =
{
//  SMF-related classes first and the MIT-top system
71
{ "top",
   Top::getClassInfo,
   Top::initialiseClass },

{ "system",
   System::getClassInfo,
   System::initialiseClass },

{ "final",                          →
   final::getClassInfo,             → DEBEMOS ADICIONAR ESTAS LINEA
   final::initialiseClass },        →
```

Figura #51C Inicialización de la Clase.

Estos pasos anteriores pueden ser observados también en el archivo finalagente.h que se encuentra en el Anexo D.

5.3.2.3 Creación del Makefile: Continuando con la creación de "*scripts*" para hacer más portable y eficiente la compilación, debemos nuevamente usar un archivo make que nos permita asegurar la ejecución correcta de la utilidad make de Linux, la ejecución del CONFIG.make y nuestro Makefile para crear el agente, este archivo make es idéntico al usado en la creación del MO. Para lograr la creación del agente como un archivo ejecutable, el "*script*" Makefile en este caso debe incluir los caminos de las librerías que contengan todos los objetos que deseemos tales como los objetos que trae incluidos OSIMIS (libisodemib.a, libuxmib.a etc) así como objetos creados por nosotros (en este caso libfinalmib.a), y las librerías que contienen los objetos genéricos deben ser incluidas obligatoriamente (libgms.a, libmsap.a etc), además el "*script*" debe contener las opciones de limpieza de archivos, de instalación, de compilación y variables de entorno necesarias, lo anterior es aclarado con la implementación del "*script*" Makefile que se encuentra en el Anexo D.

La compilación e instalación del agente se hace muy sencillo luego de la creación del "*script*" Makefile, pues en este están incluidas las opciones de compilación e instalación necesarias; por lo tanto la compilación se puede efectuar con la siguiente línea:

./make all

lo cual creara el archivo ejecutable del agente, para nuestro caso se llamara "finalagente", y su instalación la efectuamos con:
./make install-prog

con lo cual podemos pasar a utilizar el agente luego de organizar las cuestiones de inicialización y actualización de bases de datos, lo cual veremos a continuación.

5.3.2.4 Inicialización: Como observamos en el punto 5.3.2.2 el agente utiliza una tabla que contiene la inicialización de la información de los objetos definidos en la MIB o MIBs, esta información se refiere no solo al objeto que nosotros creamos, sino también a todos los objetos que incluyen las librerías que usamos en la compilación y que por tal razón podremos usar para realizar gestión. Esta tabla que en nuestro caso llamamos "mib.init.finalagente" cumple con un formato, el cual consiste de una línea declarando la instancia MO seguida por cero o mas líneas inicializando los atributos de esta instancia (como se muestra en la Figura #52). la única inicialización obligatoria que debe tener la tabla mib.init.finalagente, y en general cualquier tabla de inicialización de este tipo, es la inicialización de la clase system, la cual como sabemos hace el papel de clase base en el árbol de gestión. La tabla mib.init.finalagente que usamos puede ser observada en la Figura #52 y en el Anexo D.

```
#############################################################################
#
# mib.init.finalagente – Tabla para la inicialización de los objetos gestionados de la MIB final
#
#############################################################################
#
# formato:
# Clase             nombre distinguido relativo          nombre distinguido del padre (si lo tiene)
# Nombre del atributo = valor del atributo (opcional)
# Nombre del atributo = valor del atributo (opcional)
# Nombre del atributo = valor del atributo (opcional)
# ...


system              systemId=hostname

final               finalId=nuevo
cadenamodificable= pues si funciona esta vaina
```

Figura #52 Archivo mib.init.finalagente.

5.3.2.5 Actualización de las bases de datos: Tal como lo discutimos en el Capitulo III en los archivos isoentities e isobjects debemos incluir los datos referentes al agente creado siguiendo el respetivo formato, estos datos a incluir son mostrados en las Figuras #53 y #54:

```
...⇒⇒⇒⇒⇒ Línea # 8
"dummy-OID"          0.0
"SMA"                1.17.5.12.0
"finalagente"        1.17.5.16.0          → DEBEMOS ADICIONAR ESTA LINEA
…
```

Figura #53 Edición del Archivo isobjects.

```
...⇒⇒⇒⇒⇒ Línea # 15
osimis  SMA          1.17.5.12.0  \
                #603/Internet=200.21.83.142+11010

osimis    finalagente          1.17.5.16.0  \                    →
                #607/Internet=200.21.83.142+11014                      →
                                                          → DEBEMOS ADICIONAR ESTAS LINEA
tao       finalagente          1.17.5.16.0  \                    →
                #607/Internet=200.21.83.185+11014                      →
...
```

Figura #54 Edición del Archivo isoentities.

5.3 USANDO LOS AGENTES.

Ahora con todo el conocimiento que hemos adquirido sobre OSIMIS podemos dedicarnos a realizar las tareas de gestión, lo cual hacemos colocando en funcionamiento los agentes de gestión que posee OSIMIS, los cuales ya hemos estudiado a fondo y son SMA, ODP e IQA y por supuesto el agente que creamos finalagente.

Como vimos anteriormente para finalagente en la sección 5.3.2.4, existe un archivo mib.init necesario para la inicialización de cada agente, estos archivos existen por defecto para los agentes SMA, ODP e IQA con los valores mostrados en las Figuras #55A, #55B y #55C, para finalagente como ya hicimos notar el archivo mib.init.finalagente debió ser creado por nosotros y puede ser observado en el Anexo D.

```
##########################################################################
#
# mib.init - Table for Initial SMA Managed Objects
#
##########################################################################


# format: 1 line per MO instance followed possibly by zero or more
#             lines initialising attributes of that instance
#
# instance line format:
# class             rdn                         parent local dn
#                                               (without the top i.e. system rdn)
# attribute line format:
# attrName=attrValue

system              systemId=hostname


#subsystem          subsystemId=4


#transportEntity    entityId=isode              subsystemId=4


uxObj1              uxObjId=test
wiseSaying=it's easy with osimis-4.0


eventForwardingDiscriminator discriminatorId=discr1
destination=c=GB@o=UCL@ou=CS@cn=SMA:kinou
discriminatorConstruct=(eventType=attributeValueChange)


#monitorMetric     scannerId=uxObjId=test
#observedObjectInstance=uxObjId=test
#observedAttributeId=nUsers
#granularityPeriod=secs:10
#severityIndicatingGaugeThreshold={Low:5 Switch:Off High:8 Switch:On}


#uxObj2            uxObjId=test2
#nUsersThld={Low:5 Switch:Off High:7 Switch:On}
```

Figura #55A Archivo mib.init.sma por Defecto.

```
##########################################################################################
#
# mib.init - Table for Initial Managed Objects
#
##########################################################################################


# format:-
#
# class             rdn                         parent local dn
```

```
#                                                    (without the top i.e. system rdn)

system              systemId=hostname

simpleStats         simpleStatsId=test
```

Figura #55B Archivo mib.init.odptest por Defecto.

```
##############################################################################
#
# mib.init - Table for Initial PMA Managed Object(s)
#
##############################################################################


# format:-
#
# class                     rdn                                parent local dn
#

system              systemId=hostname

proxySystem                 systemId=NULL

cmipsnmpProxy               cmipsnmpProxyId=NULL               systemId=NULL

snmpSecurityParameter   snmpSecurityParameterId=NULL
          systemId=NULL@cmipsnmpProxyId=NULL

# NOTE: neither the cmipsnmpProxyAgent nor the SnmpImageMO's can be listed here.
```

Figura #55C Archivo mib.init.iqa por Defecto.

Con el conocimiento que tenemos ahora sobre el papel de cada objeto y sus atributos, acciones y notificaciones podemos entonces a hacer uso de los programas ejecutables (mibdump, mset, mcreate, etc) que nos ofrece OSIMIS y que vimos en el Capitulo III para realizar la gestión, a continuación en la Tabla #29 veremos el formato que utiliza cada ejecutable, el cual es explicado ampliamente en las ayudas en línea con que cuenta OSIMIS, por lo tanto para profundizar el conocimiento de estos programas ejecutables y para aclarar cualquier duda es importante acudir a estas ayudas en línea.

NOMBRE DEL PROGRAMA	SINOPSIS
evsink	**evsink** <agent> <host> [<eventType> ...] **evsink** <agent> <host> [<filter>]
mdelete	**mdelete** <agent> <host> -c <class> [-i <instance>] [-s <scope> [<sync>]] [-f <filter>]
mcreate	**mcreate** <agent> <host> -c <class> [-i <instance> □□-s <superiorInst>] [-r referenceInst] [-a <attrType=<attrValue>] ...
maction	**maction** <agent> <host> -c <class> [-i <instance>] [-s <scope> [<sync>]] [-f <filter>] -a <actionType[=<actionValue>]
mset	**mset** <agent> <host> -c <class> [-i <instance>] [-s <scope> [<sync>]] [-f <filter>] [-w|a|r|d] <attrType[=<attrValue>] ...
mibdump	**mibdump** <agent> <host> [-c <class> [-i <instance>] [-s <scope> [<sync>]] [-f <filter>] [-a <attr> ...]]
evlog	**evlog** <agent> <host> [{<filter> □□<event> ... }] **evlog** <agent> <host> D <n> **evlog** <agent> <host> D <n> <m> **evlog** <agent> <host> S <n> <attributeName> <attributeValue>

Tabla #29 Formato de los Programas para Realizar Gestión en OSIMIS.

A continuación veremos algunos ejemplos del uso de los programas de gestión, pero para que estos trabajen correctamente debemos primero colocar en funcionamiento los agentes usando sus programas ejecutables sma, odptestsrv, iqa y finalagente para que estos reciban las peticiones desde los programas de

gestión. Estos agentes podemos utilizarlos como un demonio en segundo plano o usarlos en primer plano para observar la información de las tareas que llevan a cabo, en los ejemplos que daremos aquí, ejecutamos los agentes en primer plano para poder observar en pantalla el avance de las tareas ejecutándose, y asumimos que dos máquinas con nombres “osimis”y “tao” tienen la plataforma OSIMIS instalada, para de esta manera poder realizar las peticiones de gestión remota.

Ejemplo #1: En este ejemplo realizaremos una petición al agente IQA sin especificar ninguna clase, filtro o “scope”, por lo cual el agente asumirá que la petición se efectúa sobre el objeto base del árbol de gestión.

Terminal donde corre el Agente	**Terminal donde se hace la petición de Gestión**
Listening... CMIS association established – msd 5 Get request id 00000001 class inst{} Scope baseObject CMIS association closed – msd 5	[root@tao root]#mibdump IQA osimis tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Get Result - id 0x000001, class system, instance {}, time 01030205161645 objectClass: system nameBinding: dummy-OID systemId: localhost systemTitle: c=CO@o=UC@ou=CX@cn=IQA@systemId=localhost operationalState: enabled usageState: idle 1 replies received [root@tao root]#

Ejemplo #2: En este ejemplo pediremos la creación de un objeto al agente SMA, asignándole valores a algunos de los atributos del objeto a crear.

Terminal donde corre el Agente	Terminal donde se hace la petición de Gestión
Listening... CMIS association established – msd 5 Create result : created MO with inst {scannerId=34 MonitorMetric scannerId=34 ObjectClass=monitorMetric NameBinding= scanner-system scannerId: 34 administrativeState: unlocked granularityPeriod: secs:30 operationalState: enabled observedObjectInstance: uxObjId=test observedAttributeId: nUsers derivedGauge: 5 severityIndicatingGaugeThreshold: {Low:0 Switch:Off High:65535 Switch:Off} severityIndicatingTideMarkMax: maximum: cur 5 prev 0.00 reset 01030205153041Z severityIndicatingTideMarkMin: minimum: cur 5 prev 0.00 reset 01030205153041Z previousScanCounterValue: 0 previousScanGaugeValue: 0 counterOrGaugeDifference: False CMIS association closed – msd 5	[root@tao root]#mcreate SMA osimis –c monitorMetric -i scannerId=34 -a observedObjectInstance="uxObjid=test" -a observedAttributeId="nUsers" -a granularityPeriod="secs:30" tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Create Result - id 0x000001, class monitorMetric instance {scannerId=34} time 01030205153041 objectClass: monitorMetric nameBinding: scanner-system scannerId: 34 administrativeState: unlocked granularityPeriod: secs:30 operationalState: enabled observedObjectInstance: uxObjId=test observedAttributeId: nUsers derivedGauge: 5 severityIndicatingGaugeThreshold: {Low:0 Switch:Off High:65535 Switch:Off} severityIndicatingTideMarkMax: maximum: cur 5 prev 0.00 reset 01030205153041Z severityIndicatingTideMarkMin: minimum: cur 5 prev 0.00 reset 01030205153041Z previousScanCounterValue: 0

	previousScanGaugeValue: 0 counterOrGaugeDifference: False [root@tao root]#

Ejemplo #3: Aprovecharemos la potencialidad de la gestión OSI para realizar el montaje remoto de una unidad de CD, esto lo hacemos con maction para pedir a finalagente la ejecución de la acción “montarcd”.

Terminal donde corre el Agente	**Terminal donde se hace la petición de Gestión**
Listening... CMIS association established – msd 5 Confirmed Action request: id 00000001 class final, inst{finalId=Nuevo} Scope baseObject, action “montarcd” El valor de montje es 1 CMIS association closed – msd 5	[root@tao root]#maction finalagente osimis –c final –i finalId=nuevo –a montarcd=1 tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Action Result - id 0x000001, class final instance {finalId=nuevo} time 01030205153318 Result: -779329 1 succesful actions on managed objects [root@tao root]#

Ejemplo #4: Ahora vamos a utilizar mset aprovechando la facilidad que posee la gestión OSI de utilizar filtros para fijar en objetos atributos específicos que evalúen el filtro a verdadero.

Terminal donde corre el Agente	**Terminal donde se hace la petición de Gestión**
Listening... CMIS association established – msd 5 Confirmed Set request: id 00000001 class, inst{} Scope wholesubtree Filter (wisesaying) 2 object selected after scooping and filtering set result : dispached 2 linked replies CMIS association closed – msd 5	[root@tao root]#mset SMA osimis –s wholesubtree –f “(wisesaying)” –w wisesaying = “probando mset” tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Linked Reply Set - id 0x000001, class uxObj1, instance {uxObjId=test}, time 01030205154729 wiseSaying: probando mset Linked Reply Set - id 0x000001, class uxObj2, instance {uxObjId=test2}, time 01030205154729 wiseSaying: probando mset 2 replies received [root@tao root]#

Ejemplo #5: Utilizaremos la utilidad evsink para crear un objeto eventForwardingDiscriminator que envié la información log correspondiente a la creación de un objeto al destino deseado.

Terminal donde corre el Agente	Terminal donde se hace la petición de Gestión
Listening... CMIS association established – msd 5 Create request : id 0000001 class eventForwardingDiscriminator CMISAgent send event report on msd 5 Log::store() : /osimis/logs/SMA-osimis-log1 record 1 OK Create result : Created MO with inst {discriminatorId=1} EventForwardingDiscriminator discriminatorId=1 ObjectClass=eventForwardingDiscriminator NameBinding=discriminator-system discriminatorId = 4 discriminatorConstruct = (eventType=objectCreation) administrativeState = unlocked operationalState = enabled destination=	[root@tao root]#evsink SMA osimis –f “(eventType=objectCreation)” tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX EventReport – objectCreation Class : eventForwardingDiscriminator Inst : {discriminatorId=4} Time: 01030205155904 Report : ObjectInfo : SourceIndicator : managementOperation { objectClass : eventForwardingDiscrminator nameBinding = discriminator – system discriminatorId = 4 discriminatorConstruct = (eventType=objectCreation) administrativeState = unlocked operationalState = enabled destination= [root@tao root]#

Ejemplo #6: Haciendo uso de la utilidad evlog vamos a borrar un registro log existente desde la carpeta que los almacena.

Terminal donde corre el Agente	Terminal donde se hace la petición de Gestión
Listening... CMIS association established – msd 5 Delete request : id 0000001 class log, inst {logId=1} Scope baseObject Log::store() : /osimis/logs/SMA-osimis-log1 record 1 OK CMIS association closed – msd 5	[root@tao root]#evlog SMA osimis –d –L 1 tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Delete log Waiting… DeleteResult Class=log Name={logId=1} Time=01030205160444 done [root@tao root]#

Ejemplo #7: En este ejemplo borraremos una instancia de objeto haciendo uso de la utilidad mdelete

Terminal donde corre el Agente	Terminal donde se hace la petición de Gestión
Listening... CMIS association established – msd 5 Delete request : id 00000001 class uxObj1, inst {uxObjId=test}	[root@tao root]#mdelete SMA osimis –c uxObj1 –i uxObjId=test tailor:/osimis/etc/osimistailor oidtable: /osimis/etc/oidtable

Scope baseObject CMIS association closed – msd 5	OIDs 416: attributes 517: classes 48 GlobalDomain : C=CO@o=UC@ou=CX Delete Result - id 0x000001, class uxObj1 instance {uxObjId=test} time 01030205160937 1 managed objects deleted [root@tao root]#

CONCLUSIONES

La gestión de red se ha convertido en una necesidad a gritos de todas las compañías prestadoras de servicios, y es un ramo de la tecnología al cual le falta mucho por desarrollarse. De todo lo que abarca la gestión de red, la gestión de redes de telecomunicaciones es un aspecto fundamental para nosotros en la Facultad de Ingeniería Electrónica de la Universidad del Cauca, por tal razón consideramos un aporte significativo el estudio detallado que se realizo en el trabajo de grado sobre la plataforma OSIMIS, la cual ayudara a muchos estudiantes de la facultad a entrar en el mundo de la gestión de red de una manera practica.

Se noto de manera clara en esta monografía que la plataforma OSIMIS no es completa en cuanto al nivel de aplicación y por tanto no puede competir aun con plataformas de gestión comerciales, pero nos provee todas las herramientas necesarias para desarrollar aplicaciones bastante elaboradas como lo es por ejemplo una agradable interfaz grafica, la cual es una de las metas que pretendemos proponer para futuros proyectos que se desarrollen con la plataforma.

Una de nuestras principales intenciones en el trabajo de grado fue el conseguir la compenetración de OSIMIS dentro del proyecto GETWEB, para que sirviera como gestor de dispositivos basados en CMIP, y nos sentimos muy complacidos de haber alcanzado esta meta a cabalidad ya que es un paso mas para que el

proyecto GETWEB que es bastante ambicioso y provechoso a nivel investigativo para la Facultad pueda tener un final exitoso.

La gestión CMIP siempre se ha resaltado por lo complicada en sus estándares, y lo pesada que puede volver una red, pero nadie desconoce su potencial y efectividad, y OSIMIS es una implementación muy depurada que tiene tiempos de respuesta muy similares a los obtenidos con la gestión SNMP, con lo cual no ralentiza la red mas que otras aplicaciones de gestión SNMP comerciales conocidas, esto es algo que la hace muy superior a aplicaciones de gestión CMIP comerciales, las cuales se destacan por su lentitud en respuesta.

La creación de agentes para usar en OSIMIS no fue una tarea nada fácil, puesto que desafortunadamente no existe quien de soporte de la plataforma en la actualidad, y al colocarnos el reto de utilizar la plataforma en un sistema operativo de libre distribución como Linux fue aun mas difícil, pero al lograr culminar con éxito la tarea nos sentimos bastante satisfechos, ya que hacemos un aporte para la futura generación de estudiantes de la Facultad, dejando una herramienta de fácil acceso, de código abierto y con gran potencial por delante, para que puedan pasar de la teoría sobre gestión que es lo que hasta ahora se ha manejado a realizar practicas que les permitan conocer a fondo la gestión, ya sea realizando la simple gestión de un objeto o implementando nuevos objetos y agentes que les permitan realizar una gestión particular.

BIBLIOGRAFIA

[1] F. Barillaud, L. Deri y M. Feridun, "Network Management Using Internet Technologies", Documento der9702, http://snmp.cs.utwente.nl/bibliography/articles/general/ index.html, 1997.

[2] ITU-T M.3010, Principles for a Telecomunications Management Network, Working Part IV, Report 28, Dec. 1991.

[3] CCITT G.784, SDH "Synchronous Digital Hierarchy" standards published: 1986.

[4] Martin de Pricker. Asynchronous Transfer Mode. 2nd. ed. Ellis Horwood.

[5] ITU-T X.701, Information Technology - Open Systems Interconnection – System Management Overview, July 1991

[6] ITU-T Rec X.710. " Information Technology – Open Systema Interaction – Common Management Information Service Definition". V,2. 1991.

[7] ITU-T Rec X.711. " Information Technology – Open Systema Interaction – Common Management Information Protocol Specification". V,2. 1991.

[8] G. Pavlou. "Implementation Management OSI". Tutorial presentado a la 3ra. IFIP/IEEE. San Francisco. 1993. ftp://cs.ucl.ac.uk/osimis/tutorial-isinm93.ps.Z

[9] J. L. Arciniegas, F. Velez, “GETweb - Gestión Integrada de Telecomunicaciones soportada en el Web”, Grupo de Investigacion en Telematica, Universidad del Cauca, Octubre 1998.

[10] Object Management Group, The Common Object Request Broker Architecture and Specification (CORBA), 1991.

[11] M.T. Rose, J.P. Onions, C.J. Robbins, " The ISO Development Environment User's Manual Version 7.0" PSI Inc/X-Tel Services Ltd., July 1991.

[12] ITU-T X.701, Information Technology - Open Systems Interconnection -Systems Management Overview, July 1991.

[13]Consortium ISODE, “ISODE Manual Version 1.0 ”, Volume 7: User’s Guide, February 1993.

[14] ITU-T X.722, Information Technology - Open Systems Interconnection - Structure of Management Information. - Part 4: Guidelines for the Definition of Managed Objects, August 1991.

[16] ITU-T X.500, Information Processing - Open Systems Interconnection - The Directory; Overview of Concepts, Model and Service, 1988.

[17] Information Processing - Open Systems Interconection - File Transfer, Acces and Management, 1988.

[18] D. Cass, M. Rose, ISO Transport Service on top of TCP, Network Working Group, Request For Comments 1006, May 1987.

[19] M. Rose, ISO Presentation Services on top of TCP/IP-based internets, Network Working Group, Request For Comments 1085, December 1988.

[20] X/Open, OSI-Abstract-Data Manipulation and Management Protocols Specification, January 1992.

[21] Garlock Bruce, “RedHat RPM Source Packages”, Documento rpm-bg, http://www.aplawrence.com/Linux.html, 1997.

[22] ITU-T X.213, Information Technology - Open Systems Interconnection - Definition of Network Service, November 1995.

[23] ITU-T X.214, Information Technology - Open Systems Interconnection - Definition of Transport Service, November 1995.

[24] G. Pavlou, T.Tin, "OSIMIS User Manual Version 1.0 for System Version 4.0", Dept. of Computer Science, Univesity College London, December 1994.

[25] U. Warrier, L. Besaw, The Common Management Information Services and Protocol over TCP/IP (CMOT), Network Working Group, Request For Comments 1095, April 1989.

[26] ITU-T Rec. X.219 | ISO/IEC 9072-1, “ Information processing systems - Text communication - Remote operations - Part 1. Model, notation and service definition”. 1988.

[27] ITU-T Rec. X.217 | ISO/IEC 8649, “ Information technology - Open systems interconnection - Service definition for the Association Control Service Element “. 1994.

[28] ISO/IEC 11578. “Information technology -- Open Systems Interconnection -- Remote Procedure Call (RPC)”. 1996.

[29] ITU-T X.216, Information Technology - Open Systems Interconnection - Definition of Presentation Service, July 1994.

[30] ITU-T X.721, Information Technology - Open Systems Interconnection -Structure of Management Information - Part 2: Definition of Management Information, February 1992.

[31] ITU-T X.738, Information Technology - Open Systems Interconnection -Systems Management: Metric Objects and Attributes, 1994.

[32] ITU-T X.739, Information Technology - Open Systems Interconnection -Systems Management: Summarisation Function, 1994.

[33] ITU-T X.734, Information Technology - Open Systems Interconnection -Systems Management: Event Management Function, February 1992.

[34] ITU-T X.701, Information Technology - Open Systems Interconnection -Systems Management Overview, July 1991.

[35] ITU-T X.720, Information Technology - Open Systems Interconnection -Structure of Management Information - Part 1: Management Information Model, January 1992.

[36] Integrated Communications Management of Broadband Networks, David Griffin, Crete University Press, 1992.

[37] L.LaBarre (Editor), Forum 026 - Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Issue 1.0, October 1993.

[38] ITU-T X.735, Information Technology - Open Systems Interconnection -Systems

Management: Log Control Function, September 1992.

[39] Manual sobre GDBM disponible en “http://www.polarhome.com:793/manual/gdbm-1.8.0/”

[40] ITU-T X.730, Information Technology - Open Systems Interconnection -Systems Management: Object Management Function, January 1992.

[A1] SMIC escrito por Dave Perkins de Synoptics y es disponible en ftp://ftp.synoptics.com/eng/mibcompiler

[A2] Reilly, Jim, “Converter SMI to GDMO MIB “, Documento pmdb1995, www.cs.ucl.ac.uk/staff/S.Bhatti/papers/1995/isinm4/proxy/pmbd1995.ps.gz, 1995.

[A3] April Chang (Editor), Forum 028 - ISO/CCITT to Internet Management Proxy, Issue 1.0, October 1993.