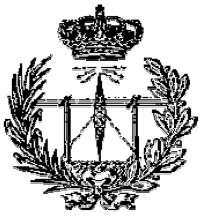
Departamento de Ingeniería de Sistemas Telemáticos

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Madrid



PROYECTO FIN DE CARRERA

Diseño e implementación de un sistema para la gestión de una aplicación distribuida de intermediación electrónica

Autor: Jorge Enrique López de Vergara Méndez

Tutor: Juan Ignacio Asensio Pérez

Madrid, septiembre de 1998.

Título: Diseño e implementación de un sistema para la gestión de una aplicación distribuida de intermediación electrónica

Autor: Jorge Enrique López de Vergara Méndez

Tutor: Juan Ignacio Asensio Pérez

Departamento: Ingeniería de Sistemas Telemáticos

Miembros del Tribunal

Presidente: Julio José Berrocal Colmenarejo

Vocal: David Fernández Cambronero

Secretario: Victor Abraham Villagrá González

Calificación: Matrícula de Honor (10 Puntos)

Fecha de lectura: 17 de septiembre de 1998.

Resumen

El presente proyecto describe el diseño e implementación de un sistema de gestión basado en SNMP para la aplicación de intermediación electrónica basada en CORBA que se está implementando en el ámbito del proyecto ACTS-ABS (*Architecture for Information Brokerage Service*, Arquitectura para un Servicio de Intermediación de Información). El sistema de gestión se basa en el uso de una pasarela SNMP-CORBA que usa los algoritmos de Traducción de Especificaciones de JIDM (*Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios) e incorpora ideas de los servicios y funcionalidades contenidos en la parte dedicada a SNMP de los documentos de Traducción de Interacción de JIDM. También se aborda la instrumentación de la información de gestión en la aplicación CORBA de forma que sea transparente a los desarrolladores de los componentes de la aplicación gestionada.

Palabras clave: Gestión integrada, CORBA, SNMP, JIDM, Java, aplicación distribuida, servicio de intermediación.

A mi familia, fuente de inspiración y apoyo.

A Alma, remanso de paz y consuelo.

Memoria

Índice

Índice de figuras

Índice de cuadros

Índice de tablas

Glosario

- **AA**: *Access Agent*, Agente de Acceso.
- **ABS**: *Architecture for a Brokerage Information Service*, Arquitectura para un Servicio de Intermediación de Información.
- **ACTS**: *Advanced Communication Technologies and Services*, Tecnologías y Servicios de Comunicación Avanzados.
- **API**: *Application Programming Interface*, Interfaz de Programación de Aplicaciones.
- **ASN.1**: *Abstract Syntax Notation 1*, Notación de Sintaxis Abstracta 1.
- **BAM**: *Broker Access Manager*, Gestor de Acceso al Intermediario.
- **BOA**: *Basic Object Adapter*, Adaptador de Objetos Básico.
- **BSA**: *Broker Service Agent*, Agente del Servicio de Intermediación.
- **BSM**. *Broker Service Manager*, Gestor del Servicio de Intermediación.
- **CMIP**: *Common Management Information Protocol*, Protocolo Común de Información de Gestión.
- **CMIS**: *Common Management Information Service*, Servicio Común de Información de Gestión.
- **CMOT**: *CMIP Over TCP*, CMIP sobre TCP.
- **CN**: *Conceptual Network*, Red Conceptual.
- **CNM**: *Conceptual Network Manager*, Gestor de la Red Conceptual.
- **CO**: *Computational Object*, Objeto Computacional.
- **CORBA**: *Common Object Request Broker Architecture*, Arquitectura Común de Intermediarios de Peticiones de Objetos.
- **COSS**: *CORBA Object Services Specification*, Especificación de Servicios de Objetos CORBA.
- **CP**: *Content Provider*, Proveedor de Contenido.
- **CPSA**: *Content Provider Service Agente*, Agente de Servicio del Proveedor de Contenido.
- **CPT**: *Content Provider Terminal*, Terminal del Proveedor de Contenido.
- **DCE**: *Distributed Computing Environment*, Entorno de Computación Distribuida.
- **DDCF**: *Distribute Document Component Facility*, Facilidad de Componentes de Documentos Distribuidos.
- **DII**: *Dinamic Invocation Interface*, Interfaz de Invocación Dinámica.

- **DNS**: *Domain Name System*, Sistema de Nombres de Dominio.
- **DPE**: *Distributed Processing Environment*, Entorno de Procesamiento Distribuido.
- **DSI**: *Dinamic Skeleton Interface*, Interfaz del Esqueleto Dinámico.
- **EFD**: *Event Forwarding Discriminator*, Discriminador de Reenvío de Eventos.
- **ESIOP**: *Environment-Specific Inter-ORB Protocols*, Protocolos Inter-ORB Específicos del Entorno.
- **FCAPS**: *Fault, Configuration, Accounting, Performance and Security*, Fallos, Configuración, Contabilidad, Rendimiento y Seguridad.
- **FM**: *Federation Manager*, Gestor de Federación.
- **GDMO**: *Guidelines for the Definition of Management Object*, Directrices para la Definición de Objetos Gestionados.
- **GIOP**: *General Interoperable Object Protocol*, Protocolo General de Interoperabilidad de Objetos.
- **GUI**: *Graphical User Interface*, Interfaz Gráfica de Usuario.
- **HEMS**: *High-level Entity Management System*, Sistema de Gestión de Entidades de Alto Nivel.
- **HTML**: *HyperText Mark-up Language*, Lenguaje de Marcas de Hipertexto.
- **HTTP**: *HyperText Transfer Protocol*, Protocolo de Transferencia de Hipertexto.
- **ICMP**: *Internet Control Message Protocol*, Protocolo de Mensajes de Control de Internet.
- **IDL**: *Interface Definition Language*, Lenguaje de Definición de Interfaces.
- **IIMC**: *ISO-Internet Management Coexistence*, Coexistencia de las Gestiones ISO e Internet.
- **IIOP**: *Internet Interoperable Object Protocol*, Protocolo de Internet de Interoperabilidad de Objetos.
- **IOR**: *Interoperable Object Reference*, Referencia de Interoperabilidad de Objetos.
- **IP**: *Internet Protocol*, Protocolo Interred.
- **IR**: *Interface Repository*, Repositorio de Interfaces.
- **ISO**: *International Standarization Organization*, Organización Internacional de Estandarización.
- **ITU**: *International Telecommunication Union*, Unión Internacional de Telecomunicación.
- **JDK**: *Java Developement Kit*, Herramienta de Desarrollo en Java.
- **JIDM**: *Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios.
- **JIDM-IT**: *JIDM – Interaction Translation*, Traducción de Interacciones de JIDM.

- **JIDM-ST**: *JIDM – Specification Translation*, Traducción de Especificaciones de JIDM.
- **LRM**: *Local Resource Manager*, Gestor de Recursos Locales.
- **MIB**: *Management Information Base*, Base de Información de Gestión.
- **MIT**: *Management Information Tree*, Árbol de Información de Gestión.
- **MSA**: *Management Service Agent*, Agente de Gestión del Servicio.
- **NMF**: *Network Management Forum*, Foro de Gestión de Redes.
- **NTP**: *Network Time Protocol*, Protocolo Horario de Red.
- **ODL**: *Object Description Language*, Lenguaje de Descripción de Objetos.
- **ODP**: *Open Distributed Processing*, Procesado Distribuido y Abierto.
- **ODP-RM**: *ODP – Refernce Model*, Modelo de Referencia ODP.
- **OID**: *Object Identifier*, Identificador de Objeto.
- **OM**: *Offer Manager*, Gestor de Ofertas.
- **OMA**: *Object Management Architecture*, Arquitectura de Gestión de Objetos
- **OMG**: *Object Management Group*, Grupo de Gestión de Objetos.
- **OMT**: *Object Modelling Technique*, Técnica de Modelado de Objetos.
- **ORB**: *Object Request Broker*, Intermediario de Peticiones de Objetos.
- **OSI**: *Open System Interconnection*, Interconexión de Sistemas Abiertos.
- **OSI-MS**: *OSI – Management System*, Sistema de Gestión OSI.
- **OT**: *Operator Terminal*, Terminal del Operador.
- **PDU**: *Protocol Data Unit*, Unidad de Datos del Protocolo.
- **PING**: *Packet Internet Grouper*, Agrupador de Paquetes de Internet.
- **PM**: *Profile Manager*, Gestor de Perfiles.
- **POA**: *Portable Object Adapter*, Adaptador de Objetos Portables.
- **QEE**: *Query Execution Engine*, Motor de Búsqueda de Consultas.
- **QoS**: *Quality of Service*, Calidad de Servicio.
- **QR**: *Query Redirector*, Redirector de Consultas.
- **RA**: *Resource Agent*, Agente de Recursos.
- **RDN**: *Relative Distinguished Name*, Nombre de Distinción Relativo.
- **RFC**: *Request for Comments*, Petición de Comentarios.
- **RM**: *Resource Manager*, Gestor de Recursos.
- **RMI**: *Remote Method Invocation*, Invocación de Métodos Remotos.
- **RPC**: *Remote Procedure Call*, Llamada a Procedimientos Remotos.

- **SGMP**: *Simple Gateway Management Protocol*, Protocolo Simple de Gestión de Pasarelas.
- **SLA**: *Service Level Agreements*, Acuerdos de Nivel de Servicio.
- **SMAE**: *System Management Application Entity*, Entidad de Aplicación de Gestión de Sistemas.
- **SMI**: *Structure of Management Information*, Estructura de la Información de Gestión.
- **SNMP**: *Simple Network Management Protocol*, Protocolo Simple de Gestión de Red.
- **SO**: *Shadow Object*, Objeto Proyectado.
- **TCL**: *Tool Command Language*, Lenguaje de Comandos de Herramientas.
- **TCP**: *Transfer Control Protocol*, Protocolo de Control de Transferencia.
- **TINA**: *Telecommunication Information Networking Architecture*, Arquitectura de Red de Información de Telecomunicación
- **TINA-C**: *TINA – Consortium*, Consorcio TINA.
- **TK**: *Tool Kit*, Juego de Herramientas.
- **TMN**: *Telecommunication Management Network*, Red de Gestión de Telecomunicaciones.
- **UML**: *Unified Modelling Language*, Lenguaje de Modelado Unificado.
- **UDP**: *User Datagram Protocol*, Protocolo de Datagramas de Usuario.
- **USA**: *User Service Agent*, Agente de Servicio del Usuario.
- **XDR**: *eXternal Data Representation*, Representación de Datos Externa.

0. Introducción

0.1. Motivación

Los marcos de gestión de red estándar *tradicionales* (tales como SNMP (*Simple Network Management Protocol*, Protocolo Simple de Gestión de Redes) [Case90], aplicado en Internet, o CMIP (*Common Management Information Protocol*, Protocolo Común de Información de Gestión) [ITUT92a], aplicado en TMN (*Telecommunication Management Network*, Red de Gestión de Telecomunicación) [ITUT92b] fueron concebidos a principios de los años 90 para la gestión de Redes de Comunicación. Fueron ellos quienes adoptaron el modelo gestor-agente, que gobierna las interacciones entre las distintas aplicaciones de gestión. Dichos modelos son orientados a objetos, pero únicamente en términos de especificación de la información que modela los recursos a gestionar y que intercambian gestor y agentes entre sí, dejando deliberadamente sin especificar aspectos que son relevantes a la estructura interna de las aplicaciones de gestión.

Por otro lado, Plataformas de Procesamiento Distribuido Orientadas a Objetos como OMG (*Object Management Group*, Grupo de Gestión de Objetos) CORBA (*Common Object Request Broker Architecture*, Arquitectura Común de Intermediación de Petición de Objetos) [OMG95a], Java RMI (*Remote Method Invocation*, Invocación de Métodos Remotos) [Sun98] y ActiveX-DCOM [Microsoft98] emergen del mundo de los sistemas distribuidos y pueden considerarse como aplicaciones prácticas del estándar de ITU/ISO referente al Marco de Referencia para Procesamiento Distribuido Abierto, (RM-ODP, *Reference Model for an Open Distributed Processing*) [ITUT95]. RM-ODP proporciona un marco de trabajo para la descripción y estandarización de sistemas de procesamiento distribuido: Sistemas de procesamiento de información cuyos componentes pueden estar situados en diferentes lugares y cuya comunicación está sujeta a posibles fallos o retrasos. Se puede decir que RM-ODP es una evolución del Modelo de Referencia OSI [ISO84] para abarcar todas aquellas particularidades de los sistemas de procesamiento distribuido. Dichos sistemas están enfocados hacia un paradigma orientado a objetos, en el que dichos objetos poseen una interfaz previamente definida por la que se puede acceder a los mismos, ya sea de forma local o remota, mediante protocolos de Llamadas a Procedimientos Remotos (RPC, *Remote Procedure Call*).

Dichas Plataformas de Procesamiento Distribuido están siendo cada vez más importantes en el mundo de los servicios de telecomunicación: Modularidad, independencia de la plataforma, reusabilidad, transparencia en las comunicaciones, etc. son ventajas a tener en cuenta para dichos servicios de telecomunicación. Un ejemplo de esta aplicación de los entornos de procesamiento distribuido al campo de las telecomunicaciones es la arquitectura de servicios de TINA (*Telecommunication Information Networking Architecture*, Arquitectura de Red de Información de Telecomunicación) [TINA95], un consorcio de operadoras de telecomunicación y fabricantes de equipos de telecomunicación que, desde 1993 vienen trabajando en una arquitectura software basada en la utilización de las facilidades de una plataforma de procesamiento distribuido orientado a objetos, arquitectura diseñada para facilitar la introducción rápida y flexible de nuevos servicios de telecomunicación e información.

Las aplicaciones y servicios desarrolladas para sistemas de procesamiento distribuido, en contraposición con las aplicaciones monolíticas, permiten poner en ejecución a los distintos objetos que las componen en diferentes máquinas comunicadas sobre una red. Este hecho puede suponer numerosas ventajas, pero por otro lado, supone que se va a necesitar algún sistema que se encargue de la gestión de ese conjunto de objetos distribuidos. Sumado a lo anterior, aparece un conjunto de situaciones que suponen nuevos retos a la hora de integrar la gestión tradicional de red y sistemas con la gestión de aplicaciones y servicios distribuidos: No sólo por las características especiales de dichas aplicaciones y servicios desde un punto de vista de gestión, sino porque es posible la administración de redes y sistemas basándose en entornos de procesamiento distribuido. Por tanto, se desplaza la tradicional separación entre operación y gestión a una visión unificada de ambas tareas, haciendo desaparecer en parte, el paradigma gestor-agente. Sin embargo, es necesario mantener la interoperabilidad con los sistemas de gestión existentes para asegurar una transición suave hacia los escenarios futuros. [Asensio97]

El presente proyecto trata relacionar todos los conceptos anteriormente descritos y aplicarlos conjuntamente a la **gestión integrada de una aplicación distribuida soportada por un entorno de procesamiento distribuido orientado a objeto**. En concreto, el proyecto ha consistido en el diseño e implementación de una arquitectura de gestión para un servicio de intermediación (*Brokerage*) electrónica soportado por una aplicación distribuida desarrollada utilizando las facilidades de CORBA. . Dicha aplicación ha sido desarrollada en el contexto del proyecto ACTS (*Advanced Communication Technologies and Services*, Servicios y Tecnologías de Comunicación avanzados [ACTS98]) ABS (*Architecture for*

Information Brokerage Service, Arquitectura para un Servicio de Intermediación de Información [ABS98][Asensio98a]), que tiene como objetivos el diseño, implementación y validación de prototipos de un sistema abierto de intermediación capaz de soportar diferentes áreas de negocio. Este servicio de intermediación electrónica consiste en poner en contacto Oferta y Demanda de una manera rápida, flexible y eficaz en el contexto del futuro "Mercado Global", en un escenario en el que el comercio se realiza a través de Internet. El servicio de intermediación electrónica de ABS incorpora principios contenidos en la arquitectura de servicios de TINA-C, utiliza CORBA como plataforma de procesamiento distribuido y JAVA como lenguaje de implementación. Implementar un sistema de gestión va a permitir poder tener control sobre la aplicación distribuida en lo que se refiere a fallos, configuración, contabilidad, rendimiento y seguridad, lo cual es muy importante a la hora de prestar un servicio.

0.2. Objetivos

Se plantearon los siguientes objetivos a la hora de realizar este Proyecto Fin de Carrera:

1. Estudio del estado del arte en lo que se refiere a sistemas de gestión distribuida.
2. Diseñar e implementar un sistema de gestión para una aplicación distribuida de intermediación electrónica con las siguientes funcionalidades:
   - **Comprobación de estado de los componentes del *Broker*** (intermediario electrónico): El administrador debe estar interesado en una visión general de qué componentes están lanzados y ejecutándose.
   - **Recogida de Información de Gestión del *Broker***: Esta función está relacionada con la necesidad de disponer de los valores de las variables de gestión de los diferentes componentes de la aplicación distribuida.
   - **Monitorización de Red y Sistema**: El administrador (haciendo uso del mismo conjunto de herramientas empleadas para la aplicación de gestión del *Broker*) puede estar interesado en obtener información sobre el rendimiento, configuración, etc. de los recursos de computación y comunicación del sistema (o sistemas, no hay que olvidar que se está hablando de una aplicación distribuida) sobre el que se implementa la aplicación.
   - **Envío de Notificaciones**: Los componentes de la aplicación pueden generar notificaciones de acuerdo a circunstancias o condiciones especiales (por ejemplo,

cuando arranca un componente). Estos informes o notificaciones pueden ser reenviados al Administrador o guardadas hasta un acceso posterior.

- **Control de Notificaciones**: El Administrador debe poder especificar qué clase de notificaciones pueden serle reenviadas o deben quedar almacenadas.
- **Registro de Notificaciones**: Soporte de las notificaciones para un acceso y procesamiento posterior.
- **Recogida de la Información del Registro**: El Administrador debe poder acceder a la información del registro sobre notificaciones emitidas previamente por los componentes de la aplicación.
- **Control del Registro (*Log*)**: El Administrador debe poder controlar la forma en que la información de registro se almacena y procesa.

Así mismo, se plantearon los siguientes requisitos y restricciones en el sistema:

- **Gestión Integrada** de red, sistema y aplicación. Un único operador debe poder gestionar todos estos aspectos a través de un mismo conjunto de herramientas.
- **Soportar las cinco áreas funcionales** incluidas el modelo **FCAPS** del modelo OSI-SM de gestión de red: Fallos, configuración, contabilidad, rendimiento y seguridad, [ITUT92a] pero enfocándose sobre todo en aspectos de configuración y rendimiento.
- **Reutilización** de tantos componentes existentes como sea posible durante las etapas de diseño e implementación. Por ello, se opta por adaptar al proyecto en estudio alguna plataforma de gestión que haya actualmente en el mercado.
- También hay que tener en cuenta otras **restricciones de implementación** (por ejemplo, CORBA, entorno de programación): Como se decidió que la gestión del sistema distribuido se basara en estándares tales como SNMP, el conjunto de la arquitectura de gestión ha dependido en gran medida de una pasarela de gestión. Este hecho se ha tenido muy en cuenta durante el proceso de diseño.

3. Análisis de prestaciones del diseño implementado.

0.3. Trabajo realizado

Las fases que han precedido la redacción de esta memoria han sido:

- Estudio del estado del arte sobre gestión de sistemas distribuidos.
- Análisis de la implementación CORBA de la compañía Iona Technologies (OrbixWeb versiones 2.0 y 3.0) que fue la utilizada en la realización del proyecto.

- Análisis de posibles alternativas de diseños y protocolos a elegir. Para esta fase se ha hecho uso del Lenguaje de Modelado de objetos Unificado (UML, *Unified Modelling Language*). Este lenguaje, con sintaxis gráfica, define un conjunto de diagramas útiles para el trabajo desarrollado en las fases de análisis y diseño de un sistema *software*.
- Implementación de una pasarela de gestión para permitir la interacción entre CORBA y el marco de gestión seleccionado.
- Implementación del subsistema de notificaciones.
- Adaptación como gestor del sistema de la plataforma de gestión *Scotty* [Schönwälder95], un paquete *software* que permite implementar aplicaciones de gestión específicas gracias a las extensiones que da al lenguaje interpretado TCL/TK (*Tool Command Language/Tool Kit* ,Lenguaje de Comandos de Herramientas/*Juego de Herramientas*) [Welch95].
- Análisis de las distintas posibilidades de gestión: Basada en Monitorización, en Eventos y combinada
- Refinamiento del sistema, añadiendo las funcionalidades de control de Notificaciones y Registros.
- Validación y medidas de rendimiento del subsistema de gestión.

0.4. Estructura de la memoria

La presente memoria ha sido dividida en las siguientes partes:

1. **Estado del arte**. En esta primera parte se trata de dar una visión sobre todos aquellos aspectos sobre los que se sustenta el desarrollo de este proyecto. Incluye los siguientes apartados:
   1.1. **Gestión de red**: Muestra de forma general las perspectivas, protocolos y arquitectura que existen sobre gestión de red.
   1.2. **Aplicaciones de Objetos Distribuidos**: Describe sucintamente la arquitectura que propugna OMG (*Object Management Group*, Grupo de Gestión de Objetos) para el procesamiento distribuido, así como la implementación OrbixWeb de Iona Technologies.
   1.3. **Gestión de Aplicaciones Distribuidas**: Habla del trabajo llevado a cabo para estandarizar la interoperabilidad entre los distintos protocolos de gestión y CORBA, haciendo especial hincapié en los trabajos de JIDM (*Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios) para la interoperabilidad SNMP-CORBA.

1.4. **Arquitectura para un Servicio de Intermediación de Información**: Da una visión sobre el proyecto ACTS ABS, de intermediación electrónica, cuya parte de gestión ha sido realizada en este proyecto.

2. **Desarrollo del proyecto**: Esta parte trata de explicar los métodos empleados para el análisis y diseño del sistema. Incluye:
   1.1. **Metodología empleada**: Muestra el método de desarrollo empleado, presentando el lenguaje UML.
   1.2. **Análisis del sistema de gestión de ABS**: Realiza un análisis del sistema a desarrollar.
   1.3. **Diseño del sistema de gestión de ABS**: Presenta el diseño del sistema de gestión desarrollado.
   1.4. **Implementación del sistema de gestión**: Describe superficialmente los ficheros finalmente implementados.
3. **Conclusiones y futuras líneas de trabajo**: Para finalizar la memoria se da una serie de ideas que se inducen del trabajo realizado.
4. **Referencias bibliográficas**: Se incluyen todas las referencias utilizadas durante la redacción de la memoria.

1. Estado del Arte

1.1. Gestión de red

1.1.1. Introducción

En las siguientes secciones del presente capítulo se va a tratar de presentar la forma de realizar la gestión de los recursos de comunicación de una red. El objeto primordial es tratar de dar una visión más o menos global de los cometidos típicos de los que debería hacerse cargo un sistema de gestión e introducir conceptos que han resultado clave en la implementación de este proyecto, en el cual se ampliarán integrando la gestión red con la de aplicaciones distribuidas.

Para ello se empezará dando los conceptos generales de la gestión, respondiendo a qué es y cuáles son sus tareas fundamentales, pasando por la evolución de los sistemas de gestión, para acabar viendo los modelos de gestión integrada estándar, de OSI e Internet, en base a los cuales se gestiona hoy en día las redes de telecomunicación.

1.1.2. Conceptos

Esta sección trata de presentar varios conceptos fundamentales en la gestión de red, comenzando explicando qué es y para qué sirve la gestión de red.

1.1.2.1. Definición y utilidad

La gestión de red se puede definir como el conjunto de actividades dedicadas al control y vigilancia de recursos de telecomunicación [Stallings93] [Sloman95]. El principal objetivo de la gestión es garantizar un nivel de servicio en los recursos gestionados con el mínimo coste.

Tareas típicas englobadas en lo que se conoce como gestión de red pueden ser el saber cuál es el porcentaje de utilización de la conexión al proveedor de servicios de acceso a Internet, detectar que un encaminador o *router* no funciona como debiera, cuántos ordenadores hay conectados a una red local, detectar y registrar intentos de acceso no autorizados, entre otros.

Los siguientes apartados ayudarán a responder en mayor profundidad lo que significa la gestión de red.

1.1.2.2. El paradigma gestor-agente

Una vez definido lo que es la gestión de red, conviene introducir un concepto fundamental que es compartido por la práctica totalidad de los sistemas de apoyo a la gestión que se pueden encontrar en el mercado: **El paradigma gestor-agente**.

Los componentes de una red se pueden clasificar en dos grandes grupos:

1. **Gestores**: Son los elementos de un sistema de gestión que interaccionan con los operadores humanos y desencadenan las acciones pertinentes para llevar a cabo las operaciones por ellos invocadas.
2. **Agentes**: Son los componentes de un sistema de gestión que llevan a cabo las operaciones de gestión invocadas por el gestor (o gestores) de la red.

De esta forma, los nodos de una red que posean un gestor se denominarán **nodos gestores** y los que tengan un agente de gestión recibirán el nombre de **nodos gestionados**.

La base del funcionamiento de estos sistemas de apoyo a la gestión reside en el intercambio de información de gestión entre nodos gestores y nodos gestionados. Es lo que se llama **paradigma gestor-agente**. Habitualmente, los agentes mantienen en cada nodo gestionado información acerca del estado y las características de funcionamiento de un determinado recurso de red. El gestor pide al agente que realice determinadas operaciones con esa información. Gracias a esas operaciones, el gestor podrá conocer el estado del recurso y podrá influir en su comportamiento mediante la alteración de esos datos de gestión.

El papel *pasivo* de los agentes se rompe cuando se produce alguna situación excepcional en el recurso gestionado. Es entonces cuando estos, por propia iniciativa, emiten los denominados **eventos** o **notificaciones** que son enviados a un gestor para que el sistema de gestión pueda actuar en consecuencia.

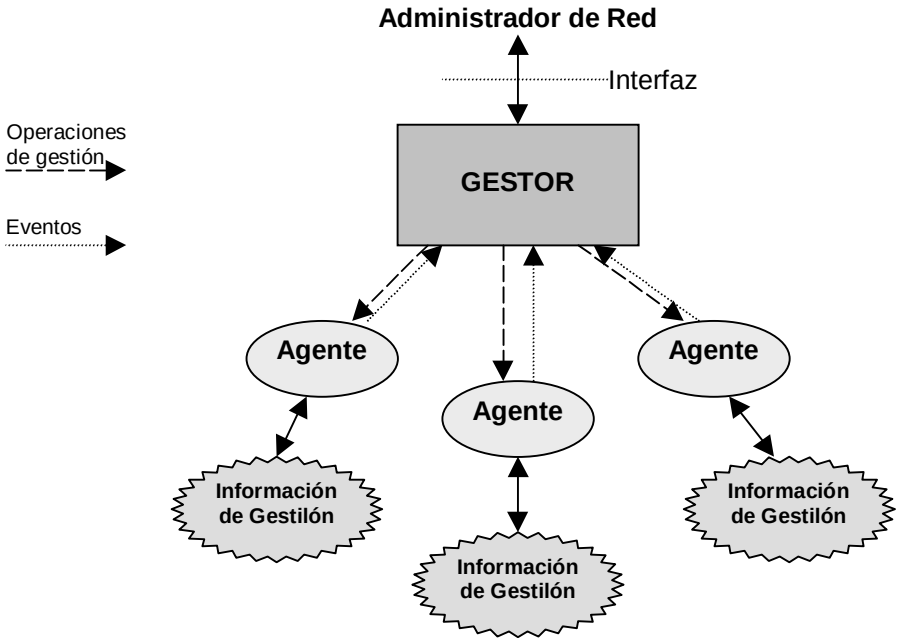


Figura 1. ***El paradigma gestor agente***

1.1.2.3. Monitorización y control

Se puede hablar de dos dimensiones presentes en la gestión de red. Una de ellas es el cómo, y está representada por la monitorización y el control. Dicha dimensión muestra cómo se realiza la gestión de los recursos. Frente a esta dimensión se encuentra el qué, representada por las áreas funcionales. Esta otra dimensión muestra qué áreas deben ser gestionadas, basándose en una división funcional.

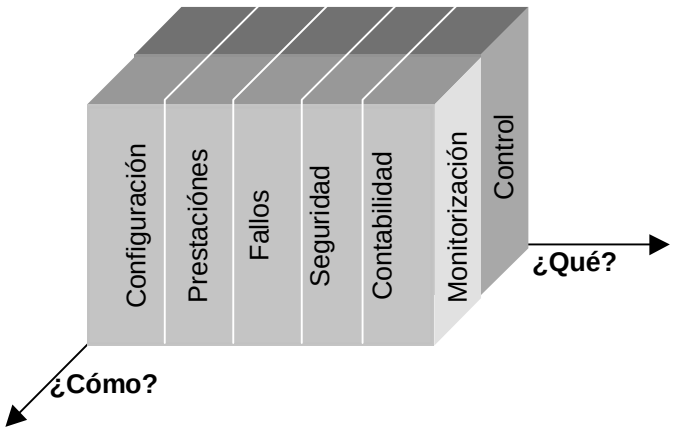


Figura 2. ***Las dimensiones de la gestión de red***

A continuación se hablará de la monitorización y el control, el cómo, presentando en el siguiente apartado las áreas funcionales, el qué.

Los sistemas de gestión han de satisfacer una gran cantidad de requisitos y llevar a buen término un sinfín de tareas. Para la consecución de todos estos objetivos, dichos sistemas se basan, en dos procedimientos básicos de actuación: la **monitorización** y el **control**. La monitorización engloba todas las operaciones de obtención de datos acerca del estado de los recursos. El procesamiento de esos datos va a permitir a los sistemas de gestión, en una fase posterior, utilizar los procedimientos de control a través de los cuáles podrán actuar sobre el comportamiento de los componentes de la red gestionada.

La **monitorización** es la parte encargada de la gestión de redes que se ocupa de la observación y análisis del estado y del comportamiento de los recursos gestionados, abarcando cuatro fases:

1. **Definición de la información** de gestión que se monitoriza. Ésta podrá ser **estática** (caracteriza la configuración de los recursos y cambia con muy poca frecuencia), **dinámica** (asociada a eventos que se dan en la red) y **estadística** (obtenida al procesar la información dinámica).

2. **Acceso a la información de monitorización**. Éste se realiza por parte de los módulos gestores a los módulos agentes localizados en los recursos mediante los denominados protocolos de intercambio de información de gestión. Éste es un punto clave en la gestión de red.

3. **Diseño de políticas de monitorización**. Se pueden distinguir dos tipos de comportamiento: **Sondeo** (el gestor juega el papel activo preguntando periódicamente a los agentes por los datos de monitorización) e **informe de eventos** (los agentes informan por propia iniciativa a los gestores ante un cambio de estado significativo)

4. **Procesado de la información de monitorización**, dándole un tratamiento que dependerá del área funcional para la que se ha realizado la monitorización.

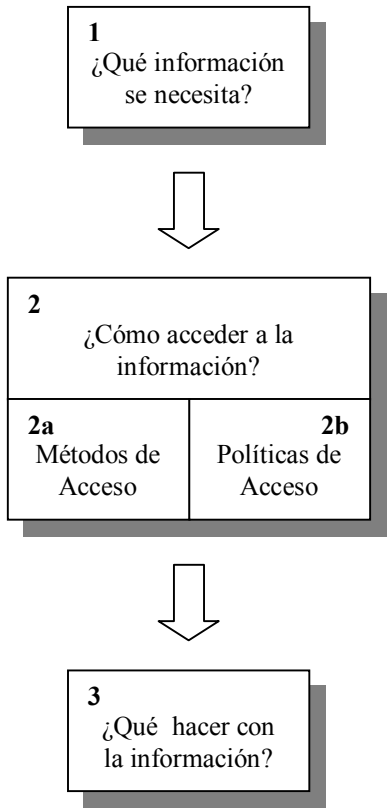


Figura 3. ***Fases en la monitorización de red***

En definitiva, implica decidir qué información se pretende conseguir, cómo se va a acceder a ella (determinando métodos y políticas de acceso) y qué se va a hacer con todos esos datos (ver figura 3)

La parte de **control** dentro de la gestión de redes es la encargada de modificar parámetros e invocar acciones en los recursos gestionados. Mientras que la monitorización es un proceso eminentemente pasivo, el control es, por contra, fundamentalmente activo.

Las tareas de control son las que más potencia aportan a los sistemas de gestión, ya que van a permitir en todo momento y de forma remota, determinar las características del comportamiento de una red.

Las funciones concretas de control están utilizadas principalmente por las áreas funcionales de configuración y seguridad.

1.1.2.4. Las áreas funcionales de gestión

Tras haber presentado en el apartado anterior la dimensión referente a cómo se gestiona, toca ahora tener en cuenta qué se gestiona. Se da a continuación la visión de las áreas funcionales que se deben tener en cuenta para realizar la gestión de los recursos de red.

A la hora de definir cuáles son las funcionalidades básicas que ha de soportar un sistema de gestión, son multitud las clasificaciones que tradicionalmente se han venido haciendo atendiendo a criterios de diversa naturaleza. Sin embargo, la ISO (*International Standarization Organization*, Organización Internacional de Estandarización) realizó una clasificación de las tareas de los sistemas de gestión en cinco **áreas funcionales**. Esta clasificación fue originariamente aplicada a los sistemas de gestión de la torre de protocolos OSI (*Open Systems Interconnection*, Interconexión de sistemas abiertos), y más tarde aplicada directa o indirectamente a la práctica totalidad de los ámbitos de gestión. A continuación se explican brevemente:

1. Gestión de **configuración**. El proceso de obtención de datos de la red y utilización de los mismos para incorporar, mantener y retirar los diferentes componentes y recursos que la integran. Tiene tres tareas fundamentales: Recolección de datos, utilización de los mismos para actuar sobre la configuración de los recursos y almacenamiento de los datos de configuración de la red.

2. Gestión de **prestaciones**. El mantenimiento del nivel de servicio que la red ofrece a sus usuarios asegurándose de que está operando de manera eficiente en todo momento, evitando situaciones de inaccesibilidad a recursos clave, y conociendo de qué forma se están empleando los recursos. Para medir todo esto hay indicadores de funcionamiento orientados al servicio y a la eficiencia.

3. Gestión de **seguridad**. Ofrecer mecanismos que faciliten el mantenimiento de políticas de seguridad, normalmente orientadas a proteger los recursos de la red de ataques llevados a cabo por intrusos. Estos podrán ser pasivos (obtención de información sin alterar datos o el funcionamiento de la red) o activos (se modifican datos, mensajes, se alteran las prestaciones de los recursos...). Para llevar a cabo la política de seguridad se suelen poner en práctica los siguientes puntos: Identificación y localización de la información a proteger, identificación de los puntos de acceso a la información, y protección y mantenimiento de dichos puntos.

4. Gestión de **fallos**. Localización y recuperación de los problemas de la red. Hay dos tareas fundamentales: Detección e identificación de los fallos (observando los síntomas, elaborando y verificando posibles hipótesis, y sacando conclusiones) y corrección del problema.

5. Gestión de **contabilidad**. Realizar estadísticas que permitan al explotador de la red generar informes de tarificación que reflejen la utilización de los recursos por parte de los usuarios. Las tareas a realizar en este punto son: Recolección de datos sobre la utilización de los recursos, establecimiento de cuotas y cobro a los usuarios de las tarifas derivadas de la utilización de los recursos.

1.1.3. Gestión Integrada

1.1.3.1. Evolución de los sistemas de gestión

La gestión de redes surgió con las redes mismas puesto que siempre ha habido necesidad de controlar, configurar, traficar, etc. los recursos de interconexión. Ahora bien, al igual que las redes han ido cambiando, los sistemas de gestión también han ido evolucionando con el paso del tiempo. Básicamente se pueden distinguir tres etapas fundamentales en la evolución de los sistemas de gestión:

1. **Gestión autónoma**: Las primeras redes tenían pocos nodos y cada uno de ellos poseía su propio sistema de gestión local. Las decisiones que afectaban a más de un nodo implicaban la comunicación con cada uno de los administradores correspondientes.
2. **Gestión homogénea**: En una etapa posterior, las redes sufrieron un aumento considerable de tamaño, pero siempre utilizando equipos y protocolos de un mismo fabricante. Ese mismo fabricante aportaba su propio sistema de gestión propietario que, en la mayoría de las ocasiones, estaba centralizado en un único nodo.
3. **Gestión heterogénea**: Más tarde, ya en nuestros días, las redes han ido creciendo y evolucionando mediante la incorporación de una amplia variedad de tecnologías. Ya no se puede hablar de entornos homogéneos sino que en una misma red se pueden encontrar componentes de una gran amalgama de fabricantes que tienen que interoperar entre sí. De esta forma, se ha conseguido aumentar los servicios ofrecidos por las redes a la vez que los explotadores de las mismas han podido maximizar el rendimiento de sus inversiones. Esta evolución de las redes ha traído consigo la necesidad de que coexistan sistemas de gestión de red de muy diversa naturaleza.

1.1.3.2. Necesidad y requisitos

La gestión heterogénea descrita en la sección anterior plantea una serie de importantes problemas desde varios puntos de vista:

1. Desde el punto de vista del usuario, es necesario que la persona o personas encargadas de la administración de la red conozcan perfectamente todos y cada uno de los sistemas que se deben utilizar.
2. Desde el punto de vista de integración de sistemas, la incompatibilidad entre datos de gestión, procedimientos, protocolos de comunicación con funcionalidad similar y que haya duplicidad y posible inconsistencia de la información almacenada en las bases de datos.

En respuesta a estas situaciones se han definido los **modelos de gestión integrada**, que permiten, teóricamente, la interconexión de una manera abierta, de los recursos de telecomunicación y las aplicaciones de gestión de red. De esa forma se podría evolucionar a modelos capaces de gestionar de una forma integrada redes heterogéneas.

Para llegar a este tipo de integración hay que tener en cuenta los dos siguientes puntos:

1. **Normalizar las comunicaciones** entre los diferentes componentes del sistema de gestión. Está claro que si un sistema de gestión quiere controlar un *router*, es necesario que éste sepa entender las preguntas que el sistema de gestión le haga con independencia del tipo de *router* y del tipo de sistema de gestión.
2. **Normalizar la información**. Este aspecto es una de las claves de los modelos de gestión de red integradas actuales y que diferencian a la gestión de red de otras aplicaciones de comunicación. El objetivo es conseguir una definición sintácticamente uniforme de todos los elementos de la red, con independencia del fabricante. Esto plantea un gran trabajo, pues es necesario realizar una definición de las propiedades de gestión de todos los recursos de comunicación existentes, y plantear la definición de la información de gestión de los recursos como una tarea más en el diseño de nuevos recursos de comunicaciones.

1.1.3.3. Modelos de gestión integrada

En la actualidad los organismos internacionales de normalización han definido tres modelos principales para la gestión de red integrada. Estos modelos serán analizados brevemente a continuación:

1. **Arquitectura TMN (*Telecomunication Management Network*, Red de Gestión de las Telecomunicaciones)**: Definida por ITU-T (*International Telecommunication Union*, Unión Internacional de las Telecomunicaciones). Más que un modelo de gestión, lo que define es una estructura de red de gestión, basada en modelos de más bajo nivel.
2. **Gestión de red OSI (*Open System Interconnection*, Interconexión de Sistemas Abiertos)**: Definido por ISO (*International Standarization Organization*, Organización de

Estandarización Internacional), con el objetivo original de lograr la gestión de los recursos de la torre de comunicaciones de OSI.

3. **Gestión Internet**: Definido por la Internet Society para gestionar los elementos de comunicaciones de las redes TCP/IP (*Transmission Control Protocol/Internet Protocol*, Protocolo de Control de Transmisión/Protocolo de Interred) [Postel81b].

1.1.4. El modelo de gestión de OSI.

1.1.4.1. Introducción

El objetivo de este apartado es tener una visión general del modelo de gestión OSI. A pesar de la poca aceptación práctica del modelo de referencia OSI, los conceptos de gestión encerrados en dicho modelo tiene plena vigencia y son la base para la comprensión de otras soluciones de gestión: La gestión OSI es una posibilidad a los sistemas de gestión integrada. Además, TMN se basa en ella para definir los modelos de información de las interfaces entre los diferentes bloques de su arquitectura. Por último, cabe decir que muchos de los puntos tratados aquí han sido tenidos en cuenta a la hora de llevar a cabo la gestión de una aplicación distribuida, tales como los discriminadores de eventos o la gestión orientada a notificaciones.

1.1.4.2. Conceptos.

El marco de gestión OSI propone tres formas distintas de llevar a cabo la gestión de entornos OSI, de manera que tengan cabida en este marco las distintas formas de realizar la gestión, incluyendo los métodos tradicionales que se seguían para acometer las tareas de gestión. Así, se definen:

1. **Gestión de Sistemas**: Pretende llevar a cabo la gestión de toda una torre de comunicaciones OSI de una manera específica, por medio de protocolos de nivel de aplicación.
2. **Gestión de nivel**: Este método de gestión OSI está pensado para permitir el intercambio de información de gestión entre elementos de red que no implementan toda la torre de protocolos OSI (una pasarela o *bridge*, un encaminador o *router*, etc.).
3. **Operación o gestión en nivel**: su objetivo es controlar y monitorizar una comunicación que atraviese un determinado nivel.

Sin embargo, restricciones tales como que la gestión OSI solo se ocuparía de recursos con capacidad de comunicación y limitarla únicamente a entornos OSI, se decidió generar una nueva norma denominada **visión general de la gestión de sistema**, definida en la norma

[ISO92], que ampliaba el ámbito de actuación y las funcionalidades de la gestión de sistemas definida en el marco de gestión OSI. La gestión de sistemas se basa en el uso de protocolos del nivel de aplicación para el intercambio de información de gestión según el paradigma gestor-agente que se define en la norma [ISO92] y explicado anteriormente. La entidad agente se encarga de aplicar operaciones actuando sobre los denominados **objetos gestionados**, que son abstracciones de los recursos controlados a través de los agentes. En los siguientes apartados se tratará de dar una visión de los modelos definidos en la gestión OSI.

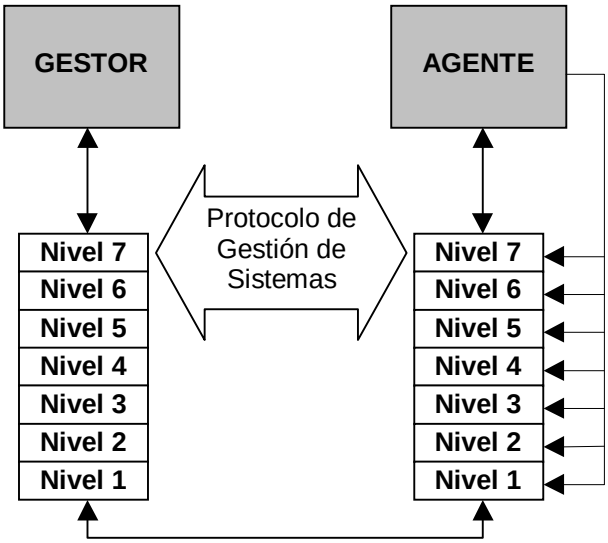


Figura 4. ***Gestión de Sistemas OSI***

1.1.4.3. Modelo funcional

Define las funciones de gestión que proporcionan servicios básicos de gestión a las aplicaciones de gestión. Dichas funciones están sujetas a normalización y se pueden encontrar en las recomendaciones [ISO91b]. Ejemplos de dichas funciones son informe de alarmas, control de registros, gestión de estados, monitorización de carga, etc.

El objetivo de estas funciones es ofrecer a las aplicaciones de gestión una funcionalidad un poco más elaboradas que el simple acceso al protocolo de gestión. Así, es posible desarrollar distintos tipos de aplicaciones de gestión que aprovechan estas funcionalidades.

1.1.4.4. Modelo de organización

Es el modelo menos tratado en la normativa sobre la gestión OSI. El objetivo de este modelo es proporcionar unas pautas para dividir las redes gestionadas en distintos dominios de gestión, basándose en cómo está configurada la red, cómo están repartidas las funcionalidades de gestión, quién tiene el papel de gestor en la red o una mezcla de los anteriores.

1.1.4.5. Modelo de comunicaciones

En el caso de la gestión de sistemas OSI se ha definido el protocolo CMIP (*Common Management Information Protocol*, Protocolo Común de Información de Gestión) (norma [ISO90]). Está situado en el nivel de aplicación de la torre OSI y proporciona el servicio denominado CMIS (*Common Management Information Service*, Servicio Común de Información de Gestión) definido en la norma [ISO91a]. Veamos a continuación las características de CMIS y CMIP:

1. **CMIS**: Se definen dos grandes tipos de servicio:

   Servicios de notificación: Se usa cuando un agente quiere informar a un gestor acerca de una notificación generada por un determinado objeto gestionado. Hay que informar del modo de conexión, del objeto que la ha generado, del tipo de notificación y otros parámetros con información adicional. Para este servicio de notificaciones va a existir una clase de objetos gestionados denominada discriminador de eventos (EFD, *Event Forwarding Discriminator*), que implementan un mecanismo de filtrado de las notificaciones que recibe y poseen un atributo que contiene la dirección del agente destino. También suele existir otro tipo de objetos que registran las notificaciones y poseen un mecanismo similar de filtrado, pero en este caso, para almacenar o no las notificaciones. El modelo de gestión OSI realiza gestión por notificaciones, y es por tanto este servicio, uno de los que más caracteriza a la gestión OSI.

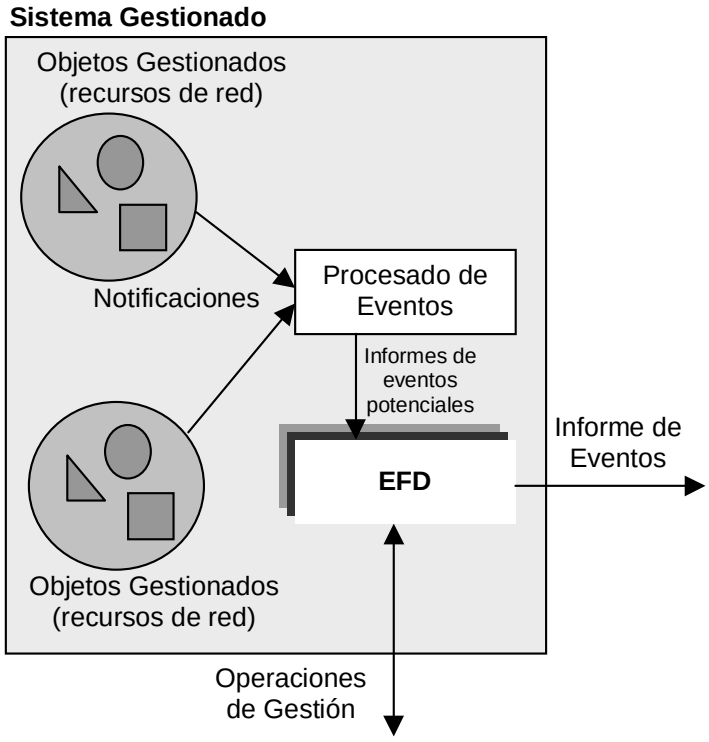


Figura 5. ***Filtrado de eventos***

Servicios de operación: Son servicios usados por el gestor para invocar operaciones de gestión a los agentes y permitir a estos devolver los resultados de dichas operaciones a los gestores. Dichos servicios son *M-GET* (pedir información de los objetos gestionados), *M-SET* (modificar o añadir información en los objetos), *M-ACTION* (llevar a cabo una acción por parte de un objeto), *M-CREATE* (crear un nuevo objeto gestionado), *M-DELETE* (eliminar un objeto gestionado) y *M-CANCEL-GET* (cancelar una petición anterior). Así mismo, se han definido una serie de métodos para poder invocar dichos servicios sobre un conjunto de objetos distintos: *Scoping* (especifica el alcance o conjunto de objetos sobre los que aplicar un filtro), filtrado (operación que se va a aplicar a los atributos de todos los objetos seleccionados por el *scoping*) y sincronización (especifica que la operación sea de tipo atómico o bien *best effort*).

2. **CMIP**: La arquitectura de este protocolo define en el nivel de aplicación una entidad de aplicación para la gestión de sistemas (SMAE, *System Management Application Entity*), que indica cómo es la configuración particular de dicho nivel en el caso de que quiera ser utilizado por una aplicación de gestión. A su vez, esta entidad define una serie de elementos para interactuar con otras entidades.

CMIP es un protocolo orientado a conexión, aportando una mayor fiabilidad pero introduciendo a la vez una sobrecarga en las comunicaciones, lo que puede no ser aceptable dado que los protocolos de gestión deben poder actuar en condiciones extremas de la red, en las que no se debe saturar más el ancho de banda con el manejo de conexiones, y que cierto tipo de gestión puede exigir una acción inmediata que no es posible si hay que establecer primeramente una conexión.

1.1.4.6. Modelo de información

La definición del modelo de información es una de las tareas más ambiciosas en la gestión de red OSI. Para poder acometer una gestión integrada, es necesario modelar de una manera común todos los recursos de comunicaciones susceptibles de ser gestionados remotamente. La norma [ISO93] hace una descripción general del modelo de información OSI, haciendo uso de los principios de la Teoría de Diseño Orientado a Objetos.

El concepto básico del modelo de información de OSI es el de **objeto gestionado**, que se define como la abstracción de recursos de comunicación o de procesado de información, con el propósito de su gestión. Este objeto se podría equiparar a una esfera opaca que rodea al recurso real y que permite la comunicación de éste último con el mundo real únicamente a través de una interfaz por la que puede intercambiar información de gestión, consistente en operaciones sobre el objeto, el resultado de dichas operaciones y notificaciones generadas por el recurso, informando de un suceso especial.

Los conceptos utilizados para la estructura de información derivan de la Teoría de Diseño Orientado a Objetos. Una vez definidas las clases de objetos gestionados es necesario poder referirse a los ejemplares de esas clases por parte del gestor. Para ello se utilizan los **nombres de ejemplares o instancias**. Esto se basa en el concepto de **jerarquía de agregación**, en la que unos objetos (objetos superiores) contienen a otros objetos (subordinados). Esta jerarquía da lugar al denominado **árbol de información de gestión** (MIT, *Management Information Tree*). Cada elemento del árbol se diferencia de los demás de su clase mediante el denominado **nombre de distinción relativo** (RDN, *Relative Distinguished Name*), que es una pareja {atributo, valor} que identifica unívocamente a ese ejemplar. Establecida la jerarquía y los RDNs, el **nombre de distinción** (DN, *Distinguished Name*) de un objeto particular se halla concatenando todos los RDN de los objetos desde la raíz al objeto en cuestión. Para especificar la organización de la jerarquía, la definición de una clase debe ir acompañada de la enumeración de las clases cuyos ejemplares pueden ser

superiores de un ejemplar de la clase que se define. Esta parte de la definición se denomina **ligadura de nombrado** (*Name Binding*).

Para realizar la definición de los objetos se usa una sintaxis basada en ASN.1 (*Abstract Syntax Notation One*, Notación de Sintaxis Abstracta Uno) que se define en la norma [ITUT94]. El lenguaje de definición de objetos gestionados es conocido como GDMO (*Guidelines for the Definition of Management Object*, Directrices para la Definición de Objetos Gestionados) en referencia al nombre de la norma en donde se define.

1.1.5. El modelo de gestión en Internet

1.1.5.1. Introducción

Internet se ha convertido en la red de ordenadores más extendida y más heterogénea de cuantas existen en la actualidad. La descentralización administrativa característica de Internet puede hacer suponer que sea inútil la utilización de sistemas de gestión para hacer frente a ese tamaño y a esa heterogeneidad.

Nada más lejos de la realidad. Al amparo de Internet ha surgido uno de los modelos de gestión integrada de más éxito de cuantos existen hoy día: El denominado genéricamente Modelo de Gestión de Internet.

Sin embargo, hay que tener en cuenta que Internet no siempre ha sido como hoy la conocemos. En un principio, el número de nodos que formaban parte de la que por aquel entonces era ARPANET (años 70) era muy reducido y la gestión se llevaba a cabo de forma autónoma (ver apartado 1.1.3.1). Posteriormente, según fue creciendo, se pasó a utilizar las facilidades ofrecidas por el protocolo ICMP (*Internet Control Message Protocol*, Protocolo de Mensajes de Control de Internet) [Postel81a], así como el PING (*Packet Internet Grouper*, Agrupador de Paquetes de Internet). Mediante este tipo de facilidades se podía hacer pruebas de conectividad, averiguar el retardo de los enlaces, etc.

A pesar de todo, a finales de los años 80, el crecimiento en tamaño y complejidad de Internet era tal que las herramientas basadas en ICMP perdieron totalmente su eficacia. Ante esto, se inició en el seno de Internet un esfuerzo investigador para desarrollar una solución de gestión integrada que pudiera enfrentarse a esa creciente complejidad adminsitrativa y de interconexión.

Así, se formaron tres grupos de trabajos, que estudiaron tres aproximaciones al problema: SGMP (*Simple Gateway Monitoring Protocol*, Protocolo Simple de Monitorización de Pasarelas) [Case87], HEMS (*High-level Entity Management System*, Sistema de Gestión de

Entidades de Alto Nivel) [Partridge87] y CMOT (*CMIP Over TCP*, CMIP Sobre TCP) [Besaw90]. Tras estos trabajos, el que tuvo más éxito fue SGMP, y tras un tiempo de mejoras, evolucionó a SNMP (*Simple Network Management Protocol*, Protocolo Simple de Gestión de Red). En su tiempo de existencia ha sufrido varias modificaciones y correcciones, y tras su estandarización en el año 90 [Case90] la más importante de ellas es la conocida como MIB-II [McCloghrie91b].

Desde el momento de la consolidación de SNMP, multitud de autores daban por hecho que, tarde o temprano, SNMP sería sustituido por otros modelos de gestión más completos que, casi con total seguridad, estarían basados, de una forma u otra, en el Marco de Gestión de Sistemas OSI. Sin embargo, lejos de desaparecer, SNMP se ha afianzado como el estándar de gestión más importante de los que se emplean en la actualidad, debido en gran medida a la proliferación de redes basadas en TCP/IP (*Transmission Control Protocol/Internet Protocol*, Protocolo de Control de la Transmisión, *Protocolo Internet*).

Los estándares en los que se basa este estándar de gestión para redes IP (*Internet Protocol*, Protocolo Interred) son las RFCs (*Request For Comments*, Peticiones de Comentarios) [Rose90] [Case90] [RFC1212] [McCloghrie91b] [Rose91b].

1.1.5.2. Axioma fundamental de la gestión

Antes de exponer la arquitectura de gestión de red utilizada en Internet, es conveniente citar el **axioma fundamental de diseño** en el que se basa:

"El impacto de añadir gestión de red a los nodos debe ser mínimo, reflejando su mínimo común denominador"

Este axioma está basado en las mismas causas que han llevado al protocolo IP a tener tan amplia aceptación: Los mínimos requisitos que impone a su capa de interfaz inferior.

Las consecuencias de este axioma son:

1. Imposibilidad de adoptar bajo este esquema el modelo de gestión de red OSI, que exige gran capacidad a los agentes para realizar operaciones tales como ámbito y filtrado sobre los objetos gestionados, aparte de tener que implementar toda la torre OSI en los agentes. Frente a ello, se ha adoptado un modelo cuya característica principal es su simpleza, que pueda implantarse en todo tipo de nodos.
2. Ausencia de una definición de modelos funcionales y de organización. Únicamente se definen modelos de comunicación e información, que serán analizados posteriormente.

1.1.5.3. Arquitectura de gestión en Internet

La arquitectura de gestión de Internet tiene una estructura mucho más sencilla que la correspondiente a los sistemas de gestión OSI. Básicamente, se identifican cuatro elementos básicos que pueden formar parte de los sistemas de gestión de Internet:

1. **Gestores**: Componentes de un sistema de gestión que invocan operaciones de gestión sobre los elementos de la red. Así mismo, interactúa con el gestor de red humano, analiza datos de gestión, recupera fallos y, en definitiva, soporta toda la inteligencia para ofrecer las capacidades típicas de un sistema de gestión.
2. **Agentes**: Componentes de un sistema de gestión que residen en los elementos gestionados y que responden a los requerimientos de información y de acciones que pide el correspondiente gestor. Los agentes pueden enviar información no solicitada de forma asíncrona.

   Existe un tipo de agente que permite la gestión de partes de la red que no comparten el modelo de gestión de Internet. Son los denominados **agentes *proxy***. Estos agentes proxy proporcionan una funcionalidad de conversión del modelo de información y del protocolo.
3. **MIB (*Management Information Base*, Base de Información de Gestión)**: Agrupación de las características actuales de todos los recursos que están bajo gestión.
4. **SNMP (*Simple Network Management Protocolo*, Protocolo Simple de Gestión de Red)**: Protocolo que permite el intercambio de información de gestión entre gestores y agentes.

Como se ve, esta arquitectura se basa en el paradigma gestor/agente, con la característica añadida de los agentes *proxy*.

1.1.5.4. Modelo de información

Al igual que en el modelo de gestión de red de OSI, el modelo de información de Internet pretende alcanzar un objetivo muy ambicioso: Elaborar unas reglas de definición de objetos, para poder representar de la misma manera todos los recursos de comunicaciones susceptible de ser gestionados.

Sin embargo, el diseño de este modelo ha sido abordado desde una perspectiva diferente: Mientras que el modelo de OSI fue diseñado para poder representar recursos complejos, y dotar a dichos recursos con parte de la funcionalidad de gestión de red, el modelo de información de la gestión de Internet se diseñó con la premisa de conservar la simplicidad de los agentes y una escasa funcionalidad de gestión.

Toda la información de gestión que se intercambia en las relaciones que se establecen entre un gestor y un agente se describe desde dos puntos de vista complementarios: Por un lado la **Estructura de la Información de Gestión** (SMI, *Structure of Management Information*) [Rose90] [McCloghrie91a], que define la estructura lógica de la información, cómo se identifica y cómo se describe, es decir, la estructura sintáctica de la comunicación. Por otro lado, la **Base de Información de Gestión** (MIB, *Management Information Base*), que utilizando la sintaxis de la SMI describe los recursos que serán objetos de las relaciones de gestión. Dicho de otro modo, proporciona la semántica de la comunicación. La primera MIB estándar se definió en [McCloghrie90] pero luego fue actualizada para constituir lo que hoy en día se conoce como MIB-II [McCloghrie91b]. También existen las denominadas MIBs privadas, como puedan ser la de *Sun* o la propia de *ABS* (proyecto que se estudiará en el cuarto capítulo), y MIBs experimentales [Rose93] [Stallings93]. En el resto del presente apartado se tratará de dar una visión resumida de la SMI.

Con el objetivo de mantener la sencillez del modelo de información, el modelo de gestión de Internet no se ciñe a las técnicas de diseño orientado a objetos. En vez de ello, se usa la idea de **tipo de objeto** (*Object-type*). Estos tipos de objetos son simples variables escalares sobre los que no se definen más operaciones que las de lectura y escritura. Obviamente, hay que utilizar algún mecanismo para identificar a los objetos. Así mismo, se necesita conocer qué valores puede poseer y qué codificación utiliza. Es decir, cada tipo de objeto debe estar caracterizado por un nombre, una sintaxis y unas reglas de codificación:

1. **Nombre**: Cada ejemplar se representa de forma única por un OID (*Object Identifier*, Identificador de Objeto) que es una secuencia de enteros obtenida al recorrer los nodos del árbol global de información, tal y como muestra la figura 6.
2. **Sintaxis**: Para indicar la estructura de los valores que están mantenidos por los objetos gestionados se utiliza una macro de ASN.1 denominada OBJECT-TYPE, definida en [Rose91] y ampliada posteriormente en [Rose91a].
3. **Reglas de codificación**: Para la transmisión de los valores de los objetos definidos mediante la SMI se aplican directamente las reglas BER (*Basic Encoding Rules*, Reglas de Codificación Básica) de codificación para ASN.1.

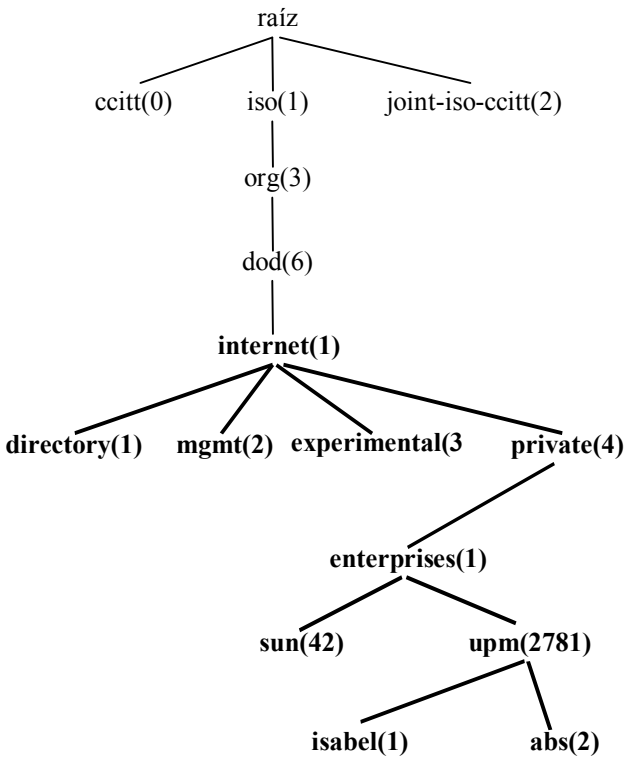


Figura 6. ***Árbol de Identificadores de Objetos***

En el siguiente cuadro se muestra un ejemplo de objetos SMI. La primera línea únicamente define un grupo de la MIB, en el que se encuentran los objetos que se definen más abajo, que son una variable simple, de tipo cadena de caracteres, accesible en lectura y escritura, y una tabla que tiene tres columnas, cada una con sus propios tipos de datos y de accesos.

```
bsm         OBJECT IDENTIFIER ::= {abs 5}
bsmLogFileName    OBJECT-TYPE
      SYNTAX      DisplayString
      ACCESS      read-write
      STATUS      mandatory
      DESCRIPTION
            "The name of the file where to log the notifications"
      ::=   {bsm 5}
bsmNotificationTable    OBJECT-TYPE
      SYNTAX      SEQUENCE OF BsmNotificationTableEntry
      ACCESS      not-accessible
      STATUS      mandatory
      DESCRIPTION
            "Information about every notifications in the last session"
```

```
      ::={bsm 6}
bsmNotificationTableEntry    OBJECT-TYPE
      SYNTAX      BsmNotificationTableEntry
      ACCESS      not-accessible
      STATUS      mandatory
      DESCRIPTION
            "Sequence of information about any notification"
      INDEX {bsmNotificationIndex}
      ::={bsmNotificationTable 1}
BsmNotificationTableEntry ::=
      SEQUENCE {
            bsmNotificationIndex          INTEGER,
            bsmNotificationTime           TimeTicks,
            bsmNotificationType           DisplayString,
      }
bsmNotificationIndex   OBJECT-TYPE
      SYNTAX      INTEGER
      ACCESS      read-only
      STATUS      mandatory
      DESCRIPTION
            "Index of BsmNotificationTable"
      ::= {bsmNotificationTableEntry 1}
bsmNotificationTime    OBJECT-TYPE
      SYNTAX      TimeTicks
      ACCESS      read-only
      STATUS      mandatory
      DESCRIPTION
            "Time from the launch when the notification was received"
      ::= {bsmNotificationTableEntry 2}
bsmNotificationType    OBJECT-TYPE
      SYNTAX      DisplayString
      ACCESS      read-only
      STATUS      mandatory
      DESCRIPTION
            "Name of the notification received"
      ::= {bsmNotificationTableEntry 3}
```

Cuadro 1. ***Ejemplo de definición SMI***

1.1.5.5. Modelo de comunicaciones

En todo modelo de gestión integrada, además de una visión común de los recursos por parte de gestores y agentes, es necesario definir un mecanismo que permita el intercambio de dicha información. En el marco de gestión de Internet dicho mecanismo consiste en el protocolo SNMP definido en la RFC 1157. En él se definen:

1. **La infraestructura de comunicación**: Uno de los requisitos de SNMP fue que **cuando el resto de la red falle, la gestión deberá continuar funcionando siempre que sea posible**. Esto provocó que funciones habituales del nivel de transporte fueran tratadas por las aplicaciones gestoras, y que el protocolo fuera no orientado a conexión. Con todo, lo normal es usar SNMP sobre UDP [Postel80] como protocolo de transporte e IP como protocolo de red. Las PDUs (*Protocol Data Units*, Unidades de datos del Protocolo) han sido definidas en ASN.1 (*Abstract Syntax Notation #1*, Notación de Sintaxis Abstracta nº 1), permitiendo así el intercambio de datos entre máquinas de arquitectura distinta.

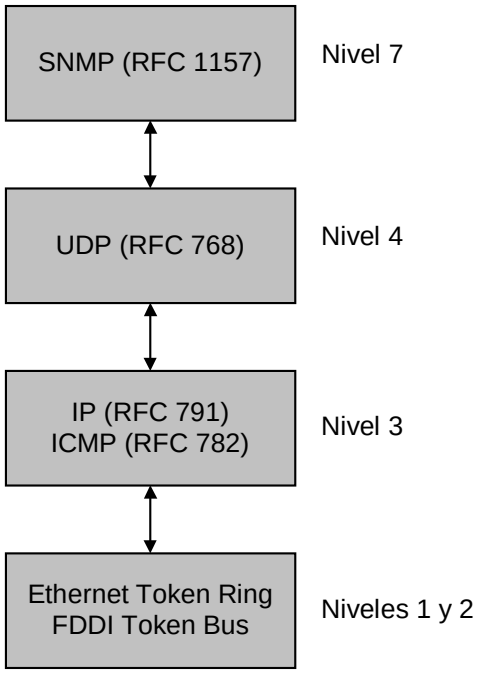


Figura 7. ***Torre de comunicaciones SNMP***

2. **Los servicios de SNMP**: Un gestor SNMP puede enviar a un agente tres tipos de mensajes: *GetRequest, GetNextRequest* y *SetRequest*. Todos ellos son confirmados por el agente mediante un mensaje del tipo *GetResponse*. Además, los agentes pueden enviar en situaciones críticas un mensaje no solicitado o *Trap* que no será confirmado. Hay que

destacar que las *Traps* SNMP no tienen la misma funcionalidad que los eventos de la gestión OSI. En SNMP se definen unos pocos *Traps* con el propósito de informar de situaciones muy específicas. Es decir: No van a permitir llevar a cabo una gestión *orientada a eventos*.

3. **Marco administrativo**: Determina las políticas de seguridad en las transacciones de gestión entre gestores y agentes. El único mecanismo de seguridad que se utiliza en la versión 1 es el basado en el concepto de **comunidad**. Una comunidad identifica una relación entre varios gestores y un agente. Los gestores pertenecientes a una comunidad poseerán unos determinados derechos de acceso a los objetos mantenidos por el agente. Cada comunidad tiene un nombre que consiste en una cadena de octetos transmitida en los mensajes SNMP. Con este nombre se puede controlar dos aspectos de seguridad distintos: **Autenticación** y **control de acceso**.

1.1.6. Conclusiones

A lo largo de este capítulo se ha descrito las funcionalidades exigibles a todo sistema de gestión que se precie, así como los modelos de gestión integrada estándar que predominan en el panorama actual dentro de la gestión de red.

Se ha podido constatar la gran importancia que tiene la consecución de una solución de gestión solución de gestión integrada eficiente para conseguir que las inversiones en telecomunicación tengan la mayor rentabilidad posible.

El modelo de gestión de mayor éxito es, y probablemente lo siga siendo durante bastante tiempo, justamente el modelo de gestión de Internet. Una vez más se demuestra que las soluciones tecnológicas aportadas por los grupos de trabajo para la estandarización de Internet destacan por su madurez, por su adecuación a la realidad del momento, por su facilidad de implementación, por su flexibilidad y por la alta relación prestación-complejidad que ofrecen [Asensio98b].

A pesar de todo ello, TMN y OSI tienen una importante presencia en las redes de los operadores de telecomunicación y, por tanto, es previsible que en el futuro todas estas soluciones de gestión deban seguir coexistiendo.

Hoy en día se plantean nuevos retos, con la aparición de las plataformas de procesamiento distribuido, que plantean la gestión, no ya solo de red o sistemas, sino de aplicaciones que corren sobre dichas plataformas. Es ésta justamente la motivación real de este proyecto: el desarrollo de un sistema de gestión basado en SNMP de una aplicación soportada por un entorno de procesamiento distribuido. En los próximos dos capítulos se

ahondará en el estado del arte de este tema, hablando primero de las plataformas de procesamiento distribuido para pasar después a su gestión.

1.2. Las aplicaciones de objetos distribuidos

1.2.1. Introducción

Tras haber estudiado en el capítulo anterior temas de gestión de red, el presente trata de presentar las aplicaciones de objetos distribuidos. Estos dos temas, que en principio tienen un contenido muy independiente, se tratarán de relacionar en el siguiente capítulo, en el que se estudia como es posible la gestión de objetos distribuido, integrándola con los marcos de gestión tradicionales.

El mundo de la orientación a objetos ha cobrado un gran impulso en los últimos tiempos al entrar en escena el procesamiento distribuido. Frente al procesamiento "monolítico", el distribuido trata de dar soluciones a problemas computacionales mediante la distribución de procesos comunicados e interactuando entre ellos.

El trabajo en aplicaciones de objetos distribuidos se ha centrado en diseñar arquitecturas de referencia y plataformas, con trasparencia de localización (un objeto invoca operaciones sobre otro sin preocuparse de dónde se encuentra). ISO elaboró el modelo de referencia ODP (*Open Distributed Processing*, Procesamiento Distribuido y Abierto) [ITUT95]. Éste se particularizó en TINA (*Telecommunication Integrated Network Architecture*, Arquitectura de Red Integrada de Telecomunicación) [TINA95] para servicios de telecomunicación.

Las implementaciones de plataformas de proceso distribuido existentes implementan subconjuntos de RM-ODP. Los más notables son: RPC [Sun88] de *Sun*, muy simple, DCE y ANSA, OLE/DCOM [Microsoft98], de *Microsoft*, y OMG-CORBA [OMG95a], estándar industrial de facto entre los proveedores de equipos y *software*.

La estructura que se va a seguir en las siguientes secciones comienza introduciendo RM-ODP, referencia para cualquier plataforma de procesamiento distribuido y TINA, arquitectura que se ha empleado en la definición de la arquitectura del presente proyecto. Siguiendo esto hay dos apartados referentes a la plataforma propuesta por OMG, para acabar presentando una implementación concreta de esta plataforma, la desarrollada por Iona Technologies [Iona].

1.2.2. RM-ODP

El modelo OSI (*Open Systems Interconnection*, Interconexión de Sistemas Abiertos) de ISO (*International Standarization Organization*, Organización Internacional de Estandarización), resuelve conceptualmente problemas de comunicación. No resuelve, sin embargo, el problema de cooperación: Capacidad de que una aplicación use los servicios de otra para proporcionar otro servicio más complejo, que se haga de forma trasparente y abierta (puedan cambiar las aplicaciones que cooperan), y que los servicios ofrecidos sean públicamente conocidos.

Para ello hace falta una capa sobre el nivel de aplicación. Es lo que se llama *middleware*, siendo esto lo que ISO trató de definir con el modelo de referencia ODP, cuyo objetivo es asegurar la interoperabilidad y ocultar la heterogeneidad a usuarios y aplicaciones.

Las principales aportaciones conceptuales del RM-ODP son la utilización del paradigma de orientación a objetos, las perspectivas o vistas y los niveles de transparencia.

En ODP se reconoce la ineficacia de describir las aplicaciones únicamente en el ámbito técnico aconsejando la inclusión de otros aspectos, proponiéndose otras **perspectivas** desde las que hay que describir un sistema:

1. **Empresa**: Contexto de negocio o político de la aplicación, pues la aplicación depende de ese entorno para el que se va a desarrollar.
2. **Información**: Modelo de objetos (OMT, UML...) de la aplicación a desarrollar o un diagrama Entidad-Relación, etc.
3. **Computación**: Conjunto de objetos computacionales que ofrecen una serie de servicios definidos mediante interfaces.
4. **Ingeniería**: Definición de los objetos de la capa *middleware* que se ocupa de la cooperación de forma transparente.
5. **Tecnología**: Opciones tecnológicas de *hardware* y *software* que hay para implementar la aplicación.

Los niveles de **transparencia** son los objetivos que ha de cumplir el modelo de ingeniería. Hay definidas trece transparencias que se deben cumplir. De ellas cabe reseñar las transparencias de:

1. **Acceso**: Un objeto puede invocar operaciones sobre otro con independencia de los mecanismos de implementación (del lenguaje en que se ha programado y el sistema operativo sobre el que se ejecuta)
2. **Localización**: Usa los servicios de otro objeto con independencia de dónde se encuentra.
3. **Migración**: Un objeto cliente no debe verse afectado por la migración del objeto servidor.
4. **Concurrencia**: Debe ser independiente del número de clientes que acceden al servicio.
5. **Fallo**: Si hay un fallo al invocar un servicio debe haber mecanismos de recuperación que hagan que el cliente no se dé cuenta. Una posible solución es mediante la replicación del servicio.
6. **Replicación**: No se debe notar la diferencia si el servidor es único o hay servidores replicados.

La transparencia de localización se consigue típicamente con RPCs (*Remote Procedure Calls*, Llamadas a Procedimientos Remotos). El cliente hace una llamada local a un *stub* o cabo (representante local del servidor en el espacio de memoria del cliente) y es el *stub* quien utiliza la infraestructura de comunicación para hablar con el *stub* (cliente ficticio) que hay en el servidor. La respuesta se realiza en el camino inverso. Los datos se transmiten con una sintaxis neutra de transmisión en línea que puede ser ASN.1 (*Abstract Syntax Notation One*, Notación de Sintaxis Abstracta Uno) o XDR (*eXternal Data Representation*, Representación de Datos Externa) para evitar problemas por distintas sintaxis. En el *marshalling* o serialización donde se produce este cambio entre sintaxis concreta de cada máquina y la neutra.

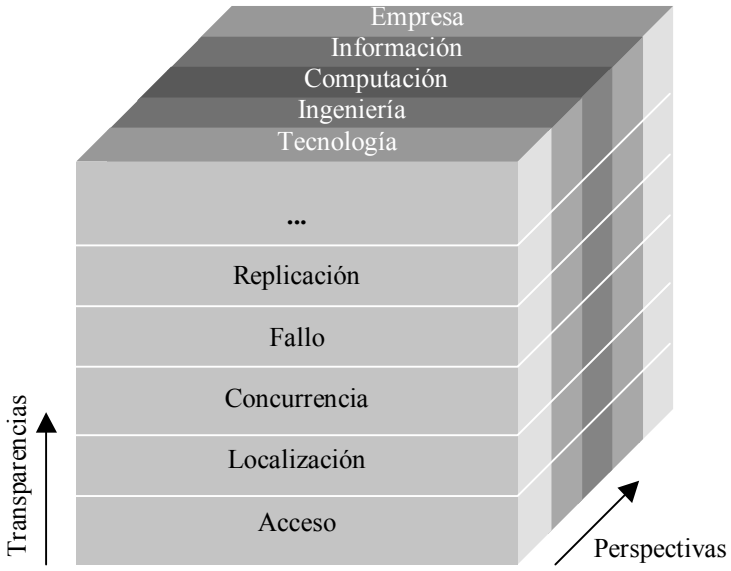


Figura 8. ***Perspectiva y Niveles de Transparencia***

1.2.3. TINA

TINA es una arquitectura *software* abierta para aplicaciones y servicios de telecomunicación. Se basa en principios de computación distribuida tales como el modelo de referencia ODP y modelado orientado a objetos. Aunque TINA define las características de su propio entorno de procesamiento distribuido (DPE, *Distributed Processing Environment*) lo que se está haciendo "de facto" para implementar aplicaciones es utilizar CORBA, añadiéndole algunas características específicas.

Su arquitectura general se puede descomponer en los siguientes nodos:

1. Arquitectura de **servicio**: Describe los modelos de sesión y de suscripción.
2. Arquitectura de **red**: Modela los recursos de red.
3. Arquitectura **computacional**: Es la arquitectura DPE.
4. Arquitectura de **gestión**: Estudia la gestión de fallos y la gestión de configuración.

Al igual que RM-ODP, TINA define perspectivas de empresa, información, computacional, ingeniería y tecnología, particularizándolas a las telecomunicaciones. En más detalle:

1. **Empresa**: Propósito, ámbito y política de un sistema para la empresa y los partícipes. Define los conceptos de s*takeholders* (actores o agentes involucrados), roles (papeles adoptados por los *stakeholders*), obligaciones (responsabilidad de los *stakeholders*) y políticas (restricciones sobre uso y operación).

2. **Información**: Marco para la especificación del modelo de información del sistema. Restricciones sobre su uso e interpretación. Define los objetos de información, los tipos o clases de objetos, las relaciones entre ellas, las restricciones y reglas (de creación, borrado, etc.) y los lenguajes (cuasi-GDMO, OMT)

3. **Computacional**: Especificación de aplicaciones distribuidas de telecomunicación en términos de Objetos Computacionales (CO, *Computational Objects*) susceptibles de distribución. Un CO es la unidad de programación y encapsulación que intercambia información a través de interfaces: Operacional, *stream* o flujo. El lenguaje utilizado es ODL (*Object Description Language*, Lenguaje de Descripción de Objetos), parecido al IDL de CORBA, que se estudiará más adelante.

4. **Ingeniería**: DPE (Entorno de Proceso Distribuido) y distribución de componentes sobre nodos de computación. Se definen los conceptos de: nodo DPE (unidad de administración de recursos en la DPE), cápsula (unidad de asignación de recursos y encapsulación), *cluster* (unidad de migración y activación). A su vez, el DPE es la suma del *kernel* o núcleo del DPE (se encarga de la creación y destrucción de objetos, y la comunicación entre ellos; hay un *kernel* en cada nodo DPE) , el KTN (*Kernel Transport Network*, Red de Transporte del Núcleo, medio de comunicación entre *kernels*) y los servicios del DPE (de nombres, de eventos, de *trading*).

1.2.4. Arquitectura OMA (*Object Management Architecture,* Arquitectura de Gestión de Objetos)

La arquitectura OMA (*Object Management Architecture*, Arquitectura de Gestión de Objetos) es la arquitectura de referencia propuesta por el consorcio industrial OMG (*Object Management Group*; Grupo de Gestión de Objetos), y de la que se deriva CORBA (*Common Object Request Broker Architecture*, Arquitectura Común de Intermediación de Peticiones de Objetos) [Schmidt97]. La arquitectura OMA pretende definir en un alto nivel de abstracción las reglas necesarias para el procesamiento distribuido orientado a objetos en entornos

heterogéneos. Trata de concebir un marco común para garantizar la interoperabilidad, mediante la integración de componentes a través de un **modelo de objetos** (*core object model*) y un **modelo de referencia**. El primer modelo define cómo se deben describir los objetos distribuidos en un entorno heterogéneo [OMG97a] y el segundo caracteriza las interacciones entre dichos objetos, estandarizando las interfaces (IDL) y protocolos (GIOP-IIOP, *General Inter-ORB Protocol - Internet Inter-ORB Protocol*, Protocolo Inter-ORB General - Protocolo Inter-ORB de Internet).

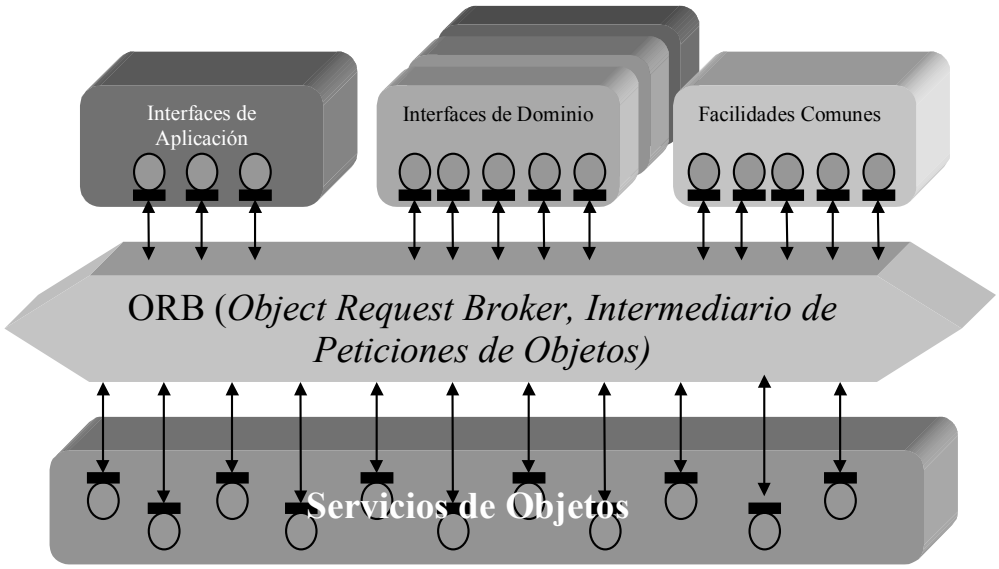


Figura 9. ***Arquitectura de Gestión de Objetos de OMG***

En el modelo de objetos de OMA un objeto es un ente encapsulado con una única identidad al que se puede acceder a sus servicios por medio de una interfaz bien definida y con transparencia de acceso y localización.

Los componentes del modelo de referencia de OMA se pueden apreciar en la figura 9. Su núcleo es el ORB (*Object Request Broker*, Intermediario de Peticiones de Objetos), especificado en CORBA, que además de dar transparencia en la localización y activación de objetos, se encarga de facilitar la comunicación entre los clientes y dichos objetos. El ORB será expuesto más adelante. A continuación se describe el conjunto de interfaces que hacen uso del ORB en el modelo de referencia en cuestión.

1.2.4.1. Servicios de objeto.

Es una colección de interfaces estandarizadas, de bajo nivel, independientes del dominio de la aplicación que las utilice debido a su uso común, que aumentan y complementan la funcionalidad del ORB. Están definidas en COSS (*Common Object Services*

Specification, Especificación de Servicios Comunes de Objetos) [OMG95b]. Entre estos servicios se puede encontrar al:

1. **Servicio de ciclo de vida**: Permite la creación, copia y portabilidad de aplicaciones.
2. **Servicio de conjunto de propiedades**: Permite a un cliente la creación, obtención y cambio de propiedades o atributos de un objeto.
3. **Servicio de nombres (*naming*)**: Permite a un cliente localizar un objeto dado su nombre. Sería como la guía telefónica de OMA. Existe una jerarquía arbórea de nombrado, al estilo de un directorio de ficheros.
4. **Servicio de t*rading***: Permite a un cliente encontrar objetos basándose en sus propiedades o su función. En este caso sería como las páginas amarillas.
5. **Servicio de eventos**: Permite una comunicación asíncrona entre objetos. Existen dos políticas de comunicación: **Eventos por *pull* (recogida)**, en que se solicita la comunicación de nuevos eventos, y **eventos por *push* (empuje)**, en que se comunica nuevos eventos sin solicitarse. A su vez, también se pueden crear canales de eventos donde se registren los objetos que quieran emitir o recibir eventos y existe la posibilidad de eventos tipados.

Hay otros tipos de servicios definidos, como el servicio de persistencia (permite a un cliente almacenar objetos persistentemente), servicio de seguridad (da entre muchas más, soporte de autenticaciones, listas de control de acceso a objetos), servicio de transacciones, de control de concurrencia, de búsqueda y de inicialización entre otros.

1.2.4.2. Facilidades comunes (*common facilities*).

Al igual que las interfaces de servicios de objetos, estas interfaces están estandarizadas, son de orientación horizontal (pueden ser usadas en cualquier campo profesional) y están orientadas a aplicaciones de usuario final. Un ejemplo puede ser la facilidad de componente de documento distribuido (DDCF, *Distributed Document Component Facility*) [Vinoski97] de OMG, basado en OpenDoc de Apple Computer, que permite la composición, presentación e intercambio de objetos basados en un modelo de documento, facilitando por ejemplo, la incorporación de un objeto hoja de calculo en un documento maestro de informes (que es igualmente un objeto) en un entorno de trabajo de personas geográficamente distribuidas. Las Facilidades comunes están en desarrollo permanente. Otros ejemplos pueden ser facilidades comunes para agentes móviles, intercambio de datos e internacionalización, etc.

1.2.4.3. Interfaces de Dominio.

Estas interfaces al igual que las mencionadas anteriormente cumplen roles similares, excepto que las del presente contexto están orientadas a aplicaciones de dominio específico, es decir, de orientación vertical. Por ejemplo, serían interfaces destinadas a aplicaciones financieras, o a aplicaciones de telecomunicaciones o de manufactura. Por ello, en la figura 9 existen diferentes gráficas de dominios de interfaces, que representa un campo o un dominio de aplicaciones (en telecomunicación, medicina, etc.).

1.2.4.4. Interfaces de Aplicación.

Aquí convergen las interfaces desarrolladas para una aplicación especifica. Estas interfaces no están estandarizadas, primero por que son aplicaciones especificas y segundo por que OMG no desarrolla, solo específica. Sin embargo, existe la posibilidad de que dichas interfaces sean candidatas a un estándar de OMG, si es que con el tiempo, algunos servicios emergen de dicha aplicación de dominio específico.

1.2.4.5 Marco de Objetos

Para finalizar esta breve introducción a OMA, cabe mencionar que existe un concepto llamado el o*bject framework* o marco de objetos. Como se muestra en la figura 10, el marco de objetos representa un conjunto de objetos que interactúan entre sí, formando una aplicación que soluciona un problema en un dominio. Cada uno de los círculos de la figura representa objetos, componentes de la aplicación, que se comunican entre sí a través de un ORB y soportan todas o algunas de las interfaces descritas en el modelo de referencia (interfaces de aplicación, interfaces de dominio especifico, facilidades comunes y/o interfaces de servicios de objeto) antes mencionado. Estos componentes se comunican entre sí (en forma *peer-to-peer* o de igual a igual), comportándose unas veces como cliente y otras como servidor.

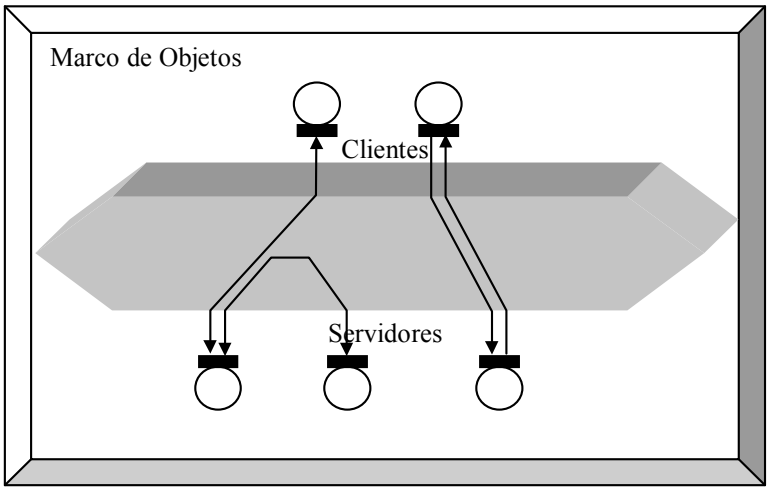


Figura 10. ***Marco de Objetos***

En la figura 10 se muestra un ejemplo de esto: Un componente realiza peticiones sobre otro componente (servidor), que a su vez, realiza más peticiones sobre otro servidor. También se ve una petición marcada como *call-back* ("contrallamada"), que ocurre cuando se recibe las respuestas mediante invocaciones del servidor (sobre el que se realizó una operación previamente) al cliente [Iona96].

1.2.5. CORBA.

CORBA (*Common Object Request Broker Architecture*, Arquitectura Común de ORBs) es una infraestructura computacional abierta de objetos distribuidos, que ha especificado la OMG con el ánimo de describir todas las características del ORB de OMA. En la figura 11 se puede apreciar cada uno de los componentes de CORBA, que serán descritos a continuación.

1.2.5.1. El cliente.

Es la entidad que invoca operaciones sobre un objeto de la implementación de objetos. Los servicios que brinda dicho objeto son transparentes al llamante. Al realizar una petición de este último sobre un objeto, ocurriendo que es remoto, se comporta como si fuese un objeto local (ver figura 11).

1.2.5.2. El ORB (*Object Request Broker*, Intermediario de Peticiones de Objetos)

El ORB es el encargado de dar **transparencia** en la comunicación a los clientes, en lo que se refiere al envío de peticiones y la devolución de respuestas ante una solicitud de servicios de un objeto. El objeto solicitado por un cliente y al que el ORB envía sus peticiones, es llamado el *target object* (objeto destino).

Mucha de la transparencia a la que antes se ha hecho alusión se refleja en que, el cliente no conoce ni necesita la localización de los objetos. Tampoco conoce la implementación de los objetos con los que desea interactuar, ni el lenguaje de programación en que están escritos, ni el sistema operativo, ni el *hardware* sobre el cual están corriendo. No es el cliente el que se preocupa de la activación de los objetos solicitados, ni tampoco de los mecanismos de comunicación (TCP/IP, llamada de métodos locales, etc.)

Por otra parte, la transparencia del ORB permite que los desarrolladores se preocupen más de sus aplicaciones y menos de los asuntos que tengan que ver con programación de sistemas distribuidos a bajo nivel.

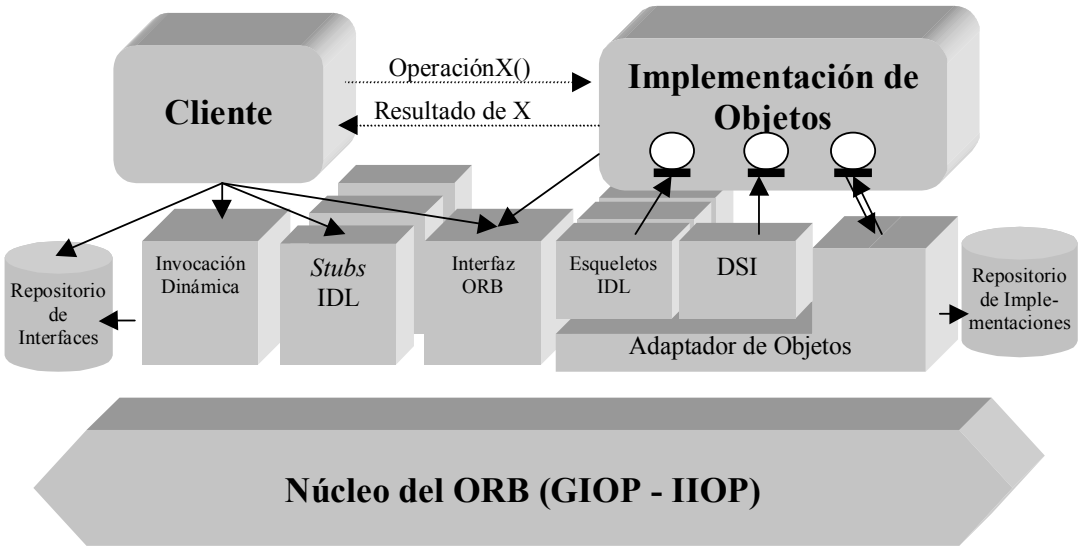


Figura 11. ***Arquitectura de CORBA***

1.2.5.3. La interfaz del ORB.

Es un conjunto de APIs (*Application Programming Interfaces*, Interfaces de Programación de Aplicaciones) que definen una serie de funciones del ORB y que pueden ser

accedidas directamente por el código cliente [OMG95a]. Entre ellas están las de convertir las referencias de objetos (información necesaria que un cliente necesita para interoperar con el ORB y dicho *target object*) en cadenas de caracteres o viceversa y las que sirven para crear estructuras de datos y listas de argumentos de peticiones, hechos a través de una invocación dinámica, que se describe más adelante.

1.2.5.4. Lenguaje de Definición de Interfaces (IDL, *Interface Definition Language*)

Cuando un cliente solicita los servicios de un objeto, este debe conocer las operaciones soportadas por dicho objeto. Las interfaces de un objeto describen dichas operaciones. Una de las ventajas de describir las operaciones de esta forma, es separar los puntos de acceso a un objeto (sus interfaces) de su propia implementación, lo que permite que los objetos sean implementados en diferentes lenguajes de programación e interactúen entre sí en forma transparente (aspecto importante en un sistema heterogéneo). Véase la figura 12.

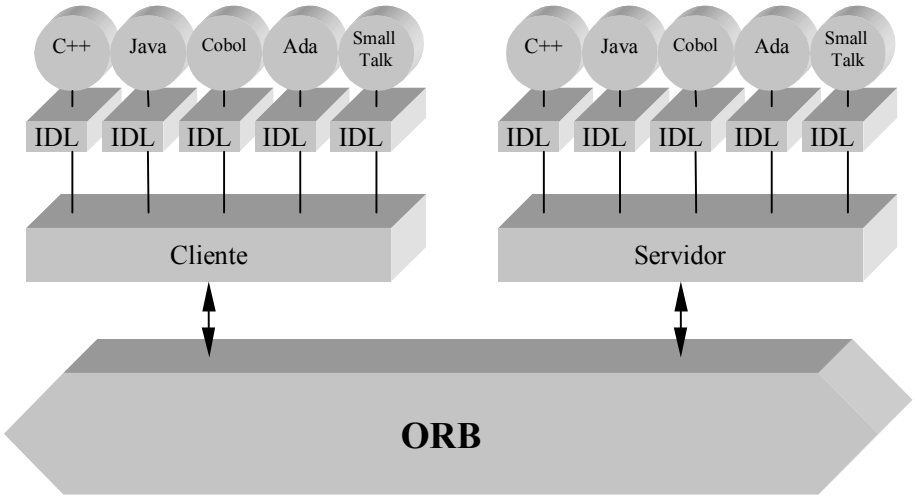


Figura 12. ***Interoperabilidad de objetos implementados en diferentes lenguajes***

La interoperabilidad de componentes CORBA se garantiza con la descripción de interfaces en lenguaje IDL. El OMG-IDL es un lenguaje de especificación parecido en estructura a C++, que permite declarar la interfaz de un objeto con el mundo exterior. Es independiente del lenguaje de implementación y plataforma de ejecución. OMG define la traducción (*mapping*) de IDL a los lenguajes más comúnmente utilizados: C, C++, Java, ADA95, Smalltalk, etc.

1.2.5.5. Cabos y esqueletos IDL

Un *stub* o cabo (también llamado *proxy*) es un ente encargado de enviar las peticiones de un cliente a un servidor a través del ORB. A esto se le denomina comúnmente *marshaling* o serialización: Se convierten las peticiones de un cliente implementado en un algún lenguaje de programación en una representación adecuada para el envío de información a través del ORB.

El *skeleton* o esqueleto es el encargado en el servidor de colaborar con la recepción de dichas peticiones desde el ORB y enviarlas a la implementación de objetos de CORBA. A esta otra operación se la llama *unmarshaling* o deserialización: Las peticiones se pasan de un formato de transmisión a un formato en un lenguaje de programación dado.

También a través del *skeleton* se envía alguna respuesta a través del ORB y es recibida por el cliente por medio del *stub*. En este caso se cambian los papeles y es el esqueleto quién serializa y el stub quién deserializa.

Se llama invocación estática al conjunto de los envíos a través del *stub* y el *skeleton*. En este caso tanto el cliente como el objeto de implementación, tienen pleno conocimiento de las interfaces IDL que están siendo invocadas.

1.2.5.6. Repositorio de interfaces (IR, *Interface Repository*)

El repositorio de interfaces es una base de datos distribuida que contiene información de las interfaces IDL definidas para los objetos que cooperarán en un entorno distribuido y que puede ser accedida o sobrescrita en tiempo de ejecución. Se puede pensar en el IR como un objeto CORBA, con una base de datos asociada y que tiene un conjunto de operaciones que se puede utilizar como si fuese un objeto cualquiera. El servicio que ofrece dicho objeto CORBA es permitir navegar sobre la jerarquía de interfaces almacenadas en la base de datos, de tal forma que se pueda saber si se quiere, la descripción de todas las operaciones que un objeto soporta [Vinoski93]. Una aplicación muy interesante y de mucha utilidad es usar el IR para descubrir interfaces de objetos en tiempo de ejecución, y acceder a ellos mediante invocación dinámica, que se verá a continuación.

1.2.5.7. Interfaz de Invocación Dinámica

La invocación dinámica es el otro tipo de invocación que existe en CORBA. Permite acceder a las operaciones de un objeto en tiempo de ejecución (*run-time*) sin tener un conocimiento previo de sus interfaces (sin un *stub*). Para dicha invocación dinámica existen dos tipos de interfaces: Una es la Interfaz de Invocación Dinámica (DII, *Dinamic Invocation*

Interface) y la otra es llamada el DSI (*Dinamic Skeleton Interface*, Interfaz Esqueleto Dinámico).

El DII es una aplicación cliente que se encarga de hacer peticiones a algún objeto del que no se conocen sus interfaces. Dicha petición se hace a través de un pseudo objeto llamado `request` (petición), sobre el cual el cliente especifica el nombre de la operación y sus argumentos, que pueden ser obtenidos del Repositorio de Interfaces (IR, *Interface Repository*). Cuando el `request` esté completo se le envía al servidor. Dicho envío puede hacerse de tres formas: En la primera el `request` se envía y todos los procesos se bloquean hasta que el servidor emita una respuesta (Invocación Sincrónica). En la segunda, cuando el `request` se envía, el cliente sigue procesando y más tarde recoge la respuesta; es la llamada Invocación Sincrónica Aplazada. En la ultima, cuando se envía el `request`, el cliente sigue procesando y la respuesta del servidor se recoge por algún otro medio, como por ejemplo, un proceso separado que la recoja.

En el lado del servidor, cuando se recibe un `request`, el DSI es quien lo toma y envía alguna respuesta al cliente ante la petición solicitada. A esto se le llama *dispatching* o despachado. En la figura 13 se puede apreciar de una manera gráfica una forma de hacer invocación dinámica.

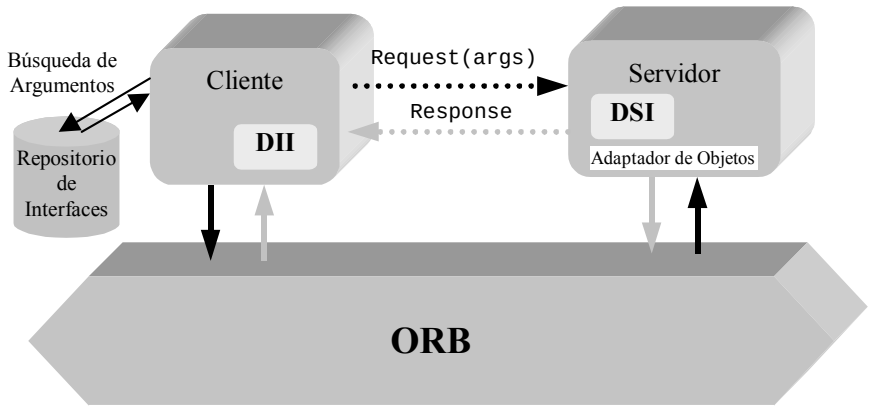


Figura 13. ***Invocación Dinámica***

1.2.5.8. Adaptador de Objetos

Es el ente de contacto entre el ORB y los objetos de implementación y es quien acepta peticiones en nombre de los objetos servidores. Dicho adaptador se encarga en tiempo de ejecución de activar, ejemplarizar, pasar peticiones y generar referencias de dichos objetos.

También, de colaborar con el ORB para que todas las peticiones que se hagan de múltiples conexiones, sean recibidas sin ningún tipo de bloqueo.

El adaptador de objetos tiene tres interfaces asociadas, una al DSI, una al IDL *skeleton* y otra a la implementación de objetos (véase figura 11) siendo las dos primeras privadas y la ultima publica, con él animo de aislar la implementación de los objetos del ORB tanto como sea posible.

La OMG ha estandarizado un adaptador de objetos llamado BOA (*Basic Object Adaptator*, Adaptador de Objetos Básico). Los servidores tienen la posibilidad de soportar más de un adaptador de objetos. OMG actualmente ha lanzado una especificación que mejora algunos defectos de portabilidad del adaptador de objetos BOA y es llamada POA (*Portable Object Adaptator*, Adaptador de Objetos Portables) [OMG97c].

1.2.5.9. Repositorio de Implementación.

Es una base de datos que, en tiempo de ejecución, da información acerca de las clases que un servidor soporta, los objetos que están ejemplarizados, sus identificadores (números únicos que asigna el adaptador de objetos a cada instancia de un objeto) y una serie de datos administrativos de los objetos, como trazas de información e información de seguridad entre otros.

1.2.5.10. Protocolos del ORB.

CORBA posee la arquitectura para posibilitar la interoperabilidad entre ORBs, con dos aproximaciones: Una directa ORB a ORB y otra basada en pasarelas (*bridges*).

El primer tipo de interoperabilidad se da cuando los ORBs residen en el mismo dominio, compartiendo las mismas referencias de objeto, el mismo tipo de información IDL y tal vez la misma información de seguridad.

El segundo tipo de interoperabilidad se da cuando se desean comunicar ORBs de diferentes dominios. Entonces el *bridge* se encarga de traducir la información específica de un ORB a otro. En la figura 14 se puede apreciar esta característica.

La arquitectura de interoperabilidad de ORBs se basa en GIOP (*General Inter-ORB Protocol*, Protocolo Inter-ORB General), que especifica un conjunto de formatos de mensajes y representaciones de datos para la interacción entre ORBs. GIOP se ha diseñado para trabajar sobre cualquier protocolo de transporte orientado a conexión. Por ejemplo, el protocolo IIOP (*Internet Inter-ORB Protocol*) especifica como se intercambia mensajes GIOP sobre redes

TCP/IP. Gracias a IIOP es posible usar Internet como un ORB *backbone* (troncal) sobre el cual, otros ORBs pueden conectarse. Para que exista compatibilidad el ORB de CORBA se debe soportar GIOP sobre TCP/IP.

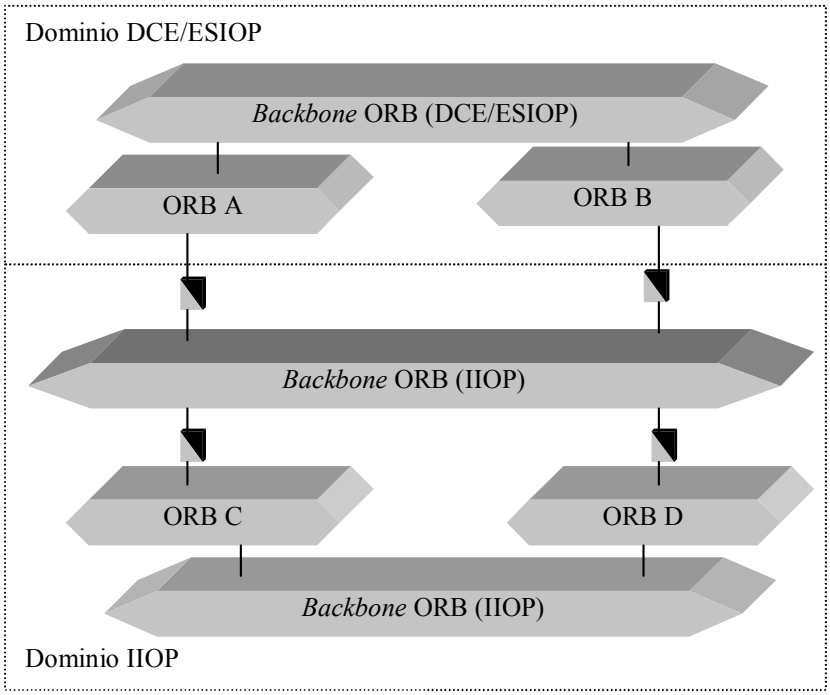


Figura 14. ***Federación de ORBs***

Por otra parte, la arquitectura de interoperabilidad entre ORBs define un conjunto de protocolos llamados ESIOPs (*Environment-specific inter-ORB Protocols*, Protocolos Inter-ORB Específicos del Entorno) que hacen posible la interacción de ORBs sobre redes especificas. Por ejemplo, uno de los primeros ESIOPs especificados fue el utilizado por el entorno de computación distribuida DCE (*Distributed Computing Environment*, Entorno de Computación Distribuida) [Rosenberry92] llamado DCE-GIOP, con él animo de que el mundo de las aplicaciones CORBA y DCE, interoperaran de forma transparente (ver figura 14).

Es importante destacar que GIOP especifica un formato para las referencias de los objetos, llamado IOR (*Interoperable Object Reference*, Referencia de Objetos Interoperables). Dicho IOR almacena la información necesaria para localizar y comunicar un objeto sobre uno o más protocolos: Datos que identifican el dominio del ORB sobre el cual se asocia una referencia y también los protocolos que ésta soporta. Por ejemplo, un IOR que contiene información de un dominio IIOP almacenará información del nombre de un *host* (nombre de una máquina o dirección IP) y un número de puerto TCP/IP.

1.2.6. La implementación de OrbixWeb

A continuación se va a presentar una de las implementaciones de CORBA más populares de las que existen en el mercado. Se trata de OrbixWeb [Iona96], creado por Iona Technologies, Inc. Al igual que la otra plataforma de esta casa, Orbix, implementa la versión 2 de CORBA, pero a diferencia de ella, el lenguaje utilizado para programar las aplicaciones que corran sobre la misma no es C++, sino Java.

El motivo de realizar esta sección no es otro que dar a conocer la implementación particular sobre la que se va a ejecutar gran parte de este proyecto, dado que fue la elegida para la realización de este proyecto, debido a su posición asentada en el mercado.

Así, en este apartado se pretende dar una visión de la arquitectura particular de OrbixWeb, comentando su funcionalidad añadida y finalmente explicando la forma en que se puede interactuar con ella programando en Java.

1.2.6.1. Arquitectura

OrbixWeb es un ORB que implementa en su totalidad la especificación CORBA 2.0. Por defecto, todos los componentes de OrbixWeb y las aplicaciones se comunican usando el protocolo estándar IIOP de CORBA.

Los componentes de OrbixWeb son los que siguen:

1. Un **compilador IDL** que analiza las definiciones IDL y produce código Java que permite desarrollar programas clientes y servidores.
2. El **OrbixWeb** en sí mismo, que es llamado por todos los programas que corren en esta plataforma y que implementa varios componentes del ORB, incluyendo la DII, la DSI, y la funcionalidad del núcleo del ORB.
3. El **demonio** (*daemon*) de OrbixWeb, que es un proceso que corre en cada máquina servidora y que implementa varios componentes del ORB, incluyendo el Repositorio de Implementaciones. También se incluye un proceso homólogo al demonio escrito en Java, que se conoce como *Java Activator* o también *orbixdj*.
4. El **Repositorio de Interfaces** de OrbixWeb, que es un proceso que implementa el Repositorio de Interfaces de CORBA.

Una versión ampliada incluye además, una implementación en Java del servicio de nombres de COSS [OMG97d]. El OrbixWeb también incluye algunas características de

programación que extienden las capacidades del ORB. Éstas se describen en la siguiente sección.

1.2.6.2. Funcionalidad añadida

Entre las características que dan un valor añadido a OrbixWeb frente a una plataforma que únicamente implemente el estándar de OMG se puede encontrar el filtrado, los *proxies* rápidos, los cargadores o los localizadores. Sin embargo, el uso de estas características supone el *software* que se desarrolle únicamente podrá correr sobre esta plataforma.

A continuación se describe someramente estas características.

1. **Filtrado**: OrbixWeb permite especificar que se ejecute código adicional antes o después del código normal de una operación o un atributo. Esta funcionalidad permite a las aplicaciones crear filtros, que pueden realizar comprobaciones de seguridad, dar trazas de depuración o simplemente información.

   Hay dos tipos de filtros: Filtros por proceso y filtros por objeto. Los **filtros por proceso** ven todas las llamadas a operaciones y atributos que entran o salen del espacio de memoria del *target object*. Los **filtros por objeto** se aplican a objetos individuales.
2. ***Smart proxies* (*proxies* rápidos)**: Permiten a los programadores implementar manualmente clases *proxy*, permitiendo, por tanto, la optimización de la interacción del cliente con servicios remotos.

   En el caso de OrbixWeb, los *stubs* son clases que se denominan *proxies*. Se generan automáticamente por el compilador IDL, y se usan para la invocación a interfaces remotas. Cuando un *proxy* recibe una invocación, la empaqueta para transmitirala al *target object* en otro espacio de memoria en la misma o en otra máquina.
3. **Cargadores (*loaders*)**: Están diseñados para el uso de objetos persistentes. Cuando una invocación de una operación llega a un proceso servidor, OrbixWeb busca el *target object* en la tabla interna de objetos en proceso. Por defecto, si el objeto no se encuentra, OrbixWeb devuelve una excepción al llamante. Sin embargo, si hay uno o más objetos cargadores instalados en el proceso, son informados sobre el fallo y se les da la oportunidad de cargar el *target object* y continuar la invocación transparentemente al llamante. Los cargadores se mantienen en una cadena, y se van probando hasta que uno puede cargar el objeto. Si ningún cargador puede cargar el objeto, se devuelve una excepción al llamante. Esto permite que los cargadores puedan dar soporte a los objetos persistentes (objetos de larga vida almacenados en disco en el sistema de ficheros o en una base de datos).

Los cargadores se pueden llamar también cuando una referencia de un objeto entra en un espacio de memoria, y no solo cuando un objeto desaparecido es destino de una petición. Esto puede surgir de varias formas: Cuando se llama a cualquiera de los métodos que permite la obtención de una referencia a un objeto dentro de un proceso. Para un servidor, como un parámetro. Para un cliente (o un servidor realizando una llamada a una operación), como un parámetro de entrada o entrada-salida, o un valor de vuelta.

Los cargadores dan la oportunidad de responder a los fallos de este tipo de objetos cargando el *target object* referenciado dentro del espacio de memoria del proceso. Si ningún cargador puede cargar el objeto referenciado, el OrbixWeb creará un *proxy* para el objeto.

4. **Localizadores**: Cuando se realiza una invocación al ORB para obtener una referencia a objeto usando un nombre de máquina anfitriona nulo, OrbixWeb usa un localizador para encontrar el *target object* en el sistema distribuido.
5. **Invocación dinámica**: En OrbixWeb se transmiten las operaciones de invocación y respuesta entre un cliente y un servidor en un objeto `org.omg.CORBA.Request`. Usando la interfaz de invocación dinámica (DII) se crea explícitamente un objeto `org.omg.CORBA.Request`, mientras que una invocación estática causa la creación implícita de dicho objeto.

   El *buffer* de datos de una petición de OrbixWeb se puede modificar permitiendo a un proceso cliente o servidor especificar que modificaciones se debe hacer a dicho *buffer* cuando una petición o una respuesta se transmite a otros procesos. La capacidad para modificar estos datos antes de su transmisión o posterior a su llegada significa que se puede añadir información adicional al flujo de datos (tal como identificación de los participantes en la comunicación), o se puede encriptar los datos como medida de seguridad o modificar de cualquier forma. La funcionalidad que se da para realizar estas transformaciones es de más bajo nivel que la que se da con los filtros, pues se permite acceder al actual buffer de datos transmitido en una petición.

Además de estas mejoras, el demonio también define una interfaz con la que se puede invocar operaciones para conocer, por ejemplo, los servidores que están actualmente en ejecución.

1.2.6.3. Java.

Java es un lenguaje de programación orientado a objetos, sintácticamente bastante parecido a C++, cuya característica más relevante es la portabilidad, es decir: tras compilar un

programa, este puede ser ejecutado en cualquier máquina, sea cual sea su arquitectura, siempre y cuando incorpore en su *software* una máquina virtual Java, que interpretará el código generado. A esta característica de independencia de la plataforma se la conoce como *write once, run everywhere* (una vez escrito, se ejecuta en cualquier parte). Dicha característica es interesante a la hora de distribuir el *software*. Además, Java abre el mundo *web* a CORBA permitiendo su integración gracias a los *applets* (aplicaciones que corren en los navegadores), que están embebidos en las páginas HTML.

Una de las principales características de OrbixWeb es que el desarrollo de las aplicaciones se realiza en Java. Se han hecho esfuerzos por compatibilizar al máximo ambas arquitecturas y se ha llegado a desarrollar el llamado CORBA/Java ORB, ya nombrado en el primer punto de este subapartado, desarrollado completamente en Java y que facilita que un *applet* o una aplicación Java (clientes) invoquen directamente métodos de objetos CORBA mediante el protocolo IIOP.

El código que se genere a partir del IDL será, por supuesto, Java y con todo ello se garantiza la portabilidad de las fuentes. Además, la versión 3.0 se ajusta a la traducción entre IDL y Java propuesta por OMG (la versión 2.0 de OrbixWeb no lo hacía). Con esto, OrbixWeb trata de poseer interoperabilidad con otras plataformas que también hagan uso de esta especificación. Esta traducción tiene un tamaño que escapa lo que aquí se quiere mostrar, pero se puede encontrar en la especificación [OMG97b].

Los pasos para crear y ejecutar una aplicación Java sobre OrbixWeb son los siguientes:

1. Se crea la definición de las clases del objeto en cuestión. Mediante IDL se puede definir las clases de los objetos, sus atributos, los métodos que exportan y los parámetros de estos.
2. Se compila el fichero IDL, creándose ficheros esqueleto de salida: Estos dependerán del número de interfaces, excepciones y estructuras de datos que se definan. Para una interfaz `*` se suele generar una serie de esqueletos:

   Del lado del cliente:

   - `*`: Es la traducción de la interfaz IDL a una interfaz Java. Define los métodos y atributos que soporta. Esta interfaz se implementa en la clase `_*Stub`.
   - `_*Stub`: Es el *proxy* que soporta las comunicaciones del lado del cliente.

   Del lado del servidor:

- `_*ImplBase`: Permite que el servidor sea implementado con el método ImplBase. Es el más sencillo, pero no permite heredar de varias interfaces distintas.
- `_*Operations`: Interfaz que traduce los atributos y métodos definidos en IDL a Java.
- `_tie_*`: Permite, junto con la interfaz anterior, que el servidor sea implementado con el método TIE. Es más complicado, pero permite heredar varias interfaces distintas.
- `_*Skeleton`: Es el *proxy* que soporta las comunicaciones del lado del servidor.

Otras clases generadas son:

- `_*Holder`: Clase que permite el paso de parámetros `out` o `inout` en operaciones IDL.
- `_*Helper`: Es una clase que contiene métodos adicionales para facilitar la labor del desarrollador.

3. Se implementan ciertos métodos descritos en estos ficheros esqueleto, con lo que crearemos las clases del servidor.
4. Se compilan los ficheros esqueleto junto con la implementación de cliente y servidor.
5. Se registran los objetos que serán ejemplarizados por el servidor en el Repositorio de Implementaciones. El ORB usa esta información para localizar objetos activos o para pedir la activación de objetos en un determinado servidor.
6. Se ejemplarizan los objetos en el servidor.
7. Se lanza el cliente.

La figura siguiente trata de mostrar la secuencia.

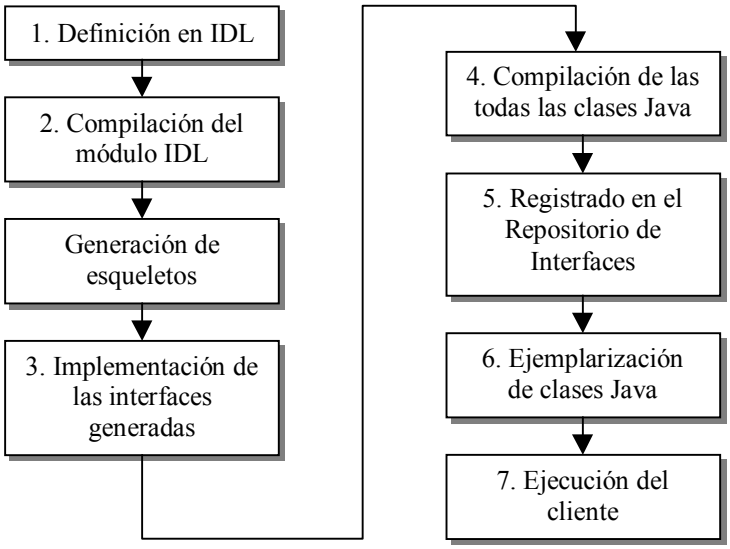


Figura 15. ***Creación de una aplicación Java***

1.2.7. Conclusiones

Los sistemas complejos y, en particular, los sistemas distribuidos y de telecomunicación cuentan con arquitecturas de referencia para facilitar su desarrollo y constituir un marco de entendimiento e interoperabilidad.

El marco de referencia conceptual para proceso distribuido y abierto es el estándar ISO-ODP. Está basado en el paradigma de orientación a objetos. TINA es una arquitectura de referencia para servicios de telecomunicación. Particulariza y extiende ODP para el desarrollo de servicios de telecomunicación. OMA-CORBA es una implementación pragmática parcialmente conforme a ODP. Propone una solución para el modelo de ingeniería y notaciones y mecanismos para construir modelos computacionales.

Gracias a los esfuerzos de la OMG, CORBA sé esta convirtiendo en un estándar de facto en el desarrollo de aplicaciones distribuidas orientadas a objetos, que permite que antiguas y nuevas aplicaciones interoperen, sin importar el entorno sobre el cual estén corriendo o el lenguaje de programación en que estén implementadas, posibilitando así, en el desarrollo de nuevas aplicaciones, la reutilización de componentes existentes.

El aspecto de modularidad que introduce CORBA permite a una aplicación aumentar su funcionalidad sin afectar al total de sus componentes, haciendo muy sencillo el desarrollo de aplicaciones a escala.

Sin embargo, el hecho de ser una nueva tecnología hace que todavía quede mucho camino por andar hasta llegar a soluciones tecnológicas óptimas. Para investigar este tema, uno de los caminos tomados ha sido el proyecto europeo ACTS-ABS, que se describe en el cuarto capítulo.

Otro problema que aparece con este avance tecnológico es la gestión de las aplicaciones distribuidas. El siguiente capítulo trata de dar una visión acerca de esta cuestión. Una aproximación a la gestión de la aplicación distribuida propuesta en ABS se estudia en la segunda parte de esta memoria.

1.3. Gestión de aplicaciones distribuidas

1.3.1. Introducción

Las aplicaciones distribuidas han creado grandes retos en el mundo de la gestión en los últimos años. Se han buscado soluciones de gestión integradas, normalizadas, escalables y efectivas para atender a la enorme proliferación y el aumento de importancia de este tipo de aplicaciones. La irrupción de plataformas de procesamiento distribuido orientadas a objetos basadas en estándares y, hasta cierto punto, especificaciones maduras tales como CORBA (*Common Object Request Broker Architecture*, Arquitectura Común de Intermediación de Peticiones de Objetos) [OMG95a], descrita en el capítulo anterior, ha dado a consorcios internacionales la oportunidad de trabajar en marcos concretos de gestión para aplicaciones distribuidas. Así, por ejemplo, ISO (*International Standard Organization*, Organización Internacional de Normalización) e ITU-T (*International Telecommunication Union,* Unión Internacional de Telecomunicaciones) están actualmente trabajando en la aplicación de los principios de ODP-RM (*Open Distributed Processing Reference Model*, Modelo de Referencia de Procesamiento Distribuido Abierto) a la gestión distribuida en ODMA (*Open Distributed Management Architecture*, Arquitectura de Gestión Distribuida Abierta) [ISO96]; TINA-C (*Telecommunication Information Networking Architecture Consortium*, Consorcio de la Arquitectura de Red de Información de Telecomunicación) ha creado recientemente un grupo de trabajo sobre gestión de servicios tomando como base su arquitectura de gestión [TINA94]; el NMF (*Network Management Forum*, Foro de Gestión de Redes) ha propuesto su Marco de Sistemas de Gestión [NMF95] y también el NMF conjuntamente con el OpenGroup han desarrollado con JIDM (*Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios) la traducción de especificaciones (JIDM-ST, JIDM *Specification Translation*) [Open97] y la traducción de interacciónes (JIDM-IT, JIDM *Interaction Translation*) [OMG98b][Mazumdar96][OMG98c], que han sido propuestas como solución para la interactuación entre TMN y CORBA por la Fuerza de Trabajos en Telecomunicación (*Telecom Task Force*) de OMG (*Object Management Group*, Grupo de Gestión de Objetos).

Estos esfuerzos se han enfocado en distintos aspectos de los problemas de gestión: Identificación y definición de las nuevas funcionalidades de gestión, adaptación de las funcionalidades de gestión existentes en el campo de la gestión de red, instrumentación de la gestión de aplicaciones distribuidas, integración de estos nuevos marcos de gestión con los ya

tradicionales, etc. Como resultado de este esfuerzo se han especificado algunos componentes de los nuevos marcos de gestión y se están refinando actualmente pero no hay una solución completa por el momento.

Como ya se ha contado en apartados anteriores, el presente proyecto trata de integrar SNMP con CORBA. Actualmente, la solución dada por los organismos internacionales pasa por los estándares de JIDM. Dado que el proyecto se ha basado en estos documentos, se les da a continuación un repaso, empezando con una visión general de lo que propone JIDM, para continuar con la traducción específica entre SNMP y CORBA.

1.3.2. JIDM

A continuación se describen los documentos desarrollados por el grupo de trabajo *Joint Inter-domain Management* (JIDM, Unión para Gestión Inter-Dominios), una actividad patrocinada conjuntamente por *The Open Group* y el *Network Management Forum* (NMF). Este proyecto se inició en respuesta a la necesidad de proporcionar herramientas que permitan la interacción entre sistemas de gestión basados en tecnologías diferentes.

En el mundo real existen varias tecnologías que son apropiadas para resolver ciertos tipos de tareas complejas. Cada una tiene fortalezas y debilidades. Pero lo que es verdaderamente claro es que tendrán en el futuro, sin duda alguna, sistemas de gestión de red. El JIDM ha identificado tres tecnologías clave, que son

- CMIP [ITUT92a],
- SNMP [Case90]**,**
- y CORBA [OMG95a],

y está tratando de crear interoperabilidad entre ellas, tanto dentro de una organización como entre organizaciones. SNMP tiene una sólida base fijada en el mercado de los ordenadores de propósito general. CMIP en el mundo de las telecomunicaciones gracias al estándar TMN, y CORBA se reconoce como el estándar emergente que cubre la programación orientada a objetos. Cada tecnología tiene sus fortalezas, con lo que una interoperabilidad completa permitiría a los diseñadores seleccionar la tecnología más apropiada para aplicar cualquier problema que se presente. El grupo *Network Management Forum ISO-Internet Management Coexistence* (IIMC, Coexistencia de Gestión ISO e Internet) [Open92] ha abordado la interoperabilidad SNMP/CMIP. Por ello, JIDM ha elegido centrarse en las interacciones CMIP/CORBA y SNMP/CORBA.

Para permitir la interacción es necesario poder hacer traducciones entre los distintos modelos de objetos y proporcionar mecanismos para manejar la conversión de protocolos en

las fronteras de cada dominio. JIDM ha obtenido resultados de comparar los modelos de objetos de la gestión OSI, CORBA y la gestión Internet. Para un par de dominios particulares, la especificación de la traducción se separa en dos partes:

1. La primera parte se refiere a la Traducción de Especificaciones, y dicta el mecanismo para hacer las traducciones entre GDMO [ITUT92c] (el lenguaje de definición de objeto que se usa con CMIP), el lenguaje de definición de las MIB SNMP [Rose90], y el Lenguaje de Definición de Interfaces (IDL, *Interface Definition Language*) de CORBA, que se usa para definir la interacción entre objetos en el dominio CORBA.
2. La segunda parte se conoce como Traducción de Interacciones, y cubre el mecanismo de conversión dinámica entre los protocolos de uno a otro dominio sin que las partes deban darse cuenta necesariamente de dicha conversión. Esto permite que se pueda representar los objetos de un dominio en otro y que las interacciones se puedan gobernar por el dominio que se quiera, en vez del dominio en el que se ha implementado un objeto en cuestión. Por ejemplo, un objeto en el dominio CORBA debe poder interactuar con un objeto GDMO como si fuera del dominio CORBA, idealmente sin que tenga que saber que este último objeto es de un dominio diferente. Naturalmente, la inversa también debe cumplirse, y un gestor OSI debe poder administrar objetos CORBA como si estuvieran definidos en GDMO (para lo cual es necesario una traducción inversa). En el caso de redes IP, también un gestor SNMP debe poder administrar objetos CORBA como si estuvieran definidos en SMI con la traducción inversa antes comentada. No se da el caso contrario, dado que SMI, debido a su simplicidad, impone muchas restricciones.

La ventaja principal de esto es que la fortaleza de CORBA (sistema orientado a objetos con interfaces bien definidas que se dirige a la comprensión y simplificación de la tarea de crear aplicaciones distribuidas) se puede combinar con la potencia de CMIP o la simplicidad de, dado que ambos son protocolos con una compatibilidad fuerte que permite la integración de *hardware* de distintos vendedores para dar lo mejor de ambos mundos, el del procesamiento distribuido y el de la gestión. El desarrollador tendría un entorno efectivo en el que implementar la funcionalidad del gestor o el agente e integrar componentes de múltiples vendedores.

1.3.2.1. Traducción de Especificaciones de JIDM (JIDM-ST)

La Traducción de Especificaciones cubre el proceso por el cual las especificaciones de información de gestión se traducen de un dominio a otro (CORBA/SNMP/CMIP). Es un proceso estático que se puede necesitar para generar material adicional durante la Traducción

de Interacciones, y que se describe mediante un conjunto de algoritmos. No se dan detalles de la información adicional requerida en la Traducción de Interacciones, pero hace referencia al proceso de la Traducción de Interacciones donde éste pueda tener importancia sobre la traducción estática.

Al traducir de GDMO a IDL, se han encontrado puntos en los que ha habido que decidir entre poder acceder a toda la potencia de CMIP y generar representaciones IDL simples que simplifiquen la tarea del programador de aplicaciones. En todos los casos que ha sido posible se ha decidido resolver el problema de acuerdo con el principio de mantener la simpleza en el mayor número de casos a expensas de hacer más complejos los menos usados.

La traducción de SMI a IDL no ha tenido mayores problemas, debido a su simpleza. Sin embargo, por esto mismo, no ha sido posible obtener una traducción de especificaciones de IDL a SMI.

El documento que describe la traducción de especificaciones es [JDIM-ST]. Dicho documento se divide en varias partes, en los que trata varios temas: Comienza estudiando el algoritmo de traducción de ASN.1 a OMG IDL, dando las reglas comunes de traducción usadas tanto en los algoritmos de traducción de GDMO como SNMP, al usar ambos para sus definiciones macros ASN.1. Posteriormente sigue con el algoritmo de traducción de GDMO a OMG IDL, abordando la traducción de especificaciones basadas en OSI-GDMO a IDL. Continúa presentando el algoritmo de traducción de OMG IDL a GDMO/ASN.1, tratando la traducción de especificaciones basadas en OMG IDL a GDMO. Otro capítulo posterior estudia el algoritmo de traducción de SNMP a OMG IDL, en donde se aborda la traducción de especificaciones basadas en SNMP a OMG IDL. Sin embargo, el algoritmo de traducción de OMG IDL a SNMP no está desarrollada por el motivo antes comentado. Además, incluye módulos IDL que están definidos en la especificación y también proporciona ejemplos informativos de la aplicación de los algoritmos definidos en ese documento. Finalmente, realiza un estudio comparando los modelos de objetos de OSI, SNMP y OMG.

Para el caso particular de la traducción de SMI a IDL, utilizada en este proyecto, se establecen las siguientes reglas, contenidas en el estándar de traducción de especificaciones de JIDM:

1. Pasar cada modulo de información SNMP a otro IDL, en el que se declara:
   - Todas las interfaces, tipos y constantes declaradas por un módulo SMI al ámbito correspondiente en un módulo IDL.
   - Los tipos importados en el módulo SMI como `typedef` de un tipo IDL importado mediante un `#include`.

- Dos interfaces IDL, llamados `SnmpNotifications` y `PullSnmpNotifications`, (N en la figura 16) si existe al menos una macro `NOTIFICATION-TYPE`. La interfaz `SnmpNotifications` se usará para comunicación de sucesos de tipo *push*, y el `PullSnmpNotifications`, para los de tipo *pull*.
- Una pseudo interfaz IDL, llamado `DefaultValues` (D en la figura 16), si hay al menos una macro `OBJECT-TYPE` con la cláusula `DEF-VAL`; las pseudo interfaces IDL generan código de biblioteca similar a la especificación de la interfaz `CORBA::TypeCode`.
- Una pseudo interfaz IDL, llamado TextualConventions (TC en la figura 16), si hay al menos una macro del tipo `TEXTUAL-CONVENTION` con la cláusula `DISPLAY-HINT`.

2. Traducir cada tipo ASN.1 en un tipo IDL, usando el esquema de traducción definido por JIDM [Open97, capítulo 2]. Un tipo de datos ASN.1 complejo (usado para describir las PDUs (*Protocol Data Units*, Unidades de Datos de Protocolo) de SNMP) puede generar más de un tipo de datos IDL.
3. Traducir el valor especificado en la cláusula `SYNTAX` de la macro `TEXTUAL-CONVENTION`, tales como `DisplayString`, en un tipo IDL:
   Si la cláusula `DISPLAY-HINT` está presente, definir dos operaciones dentro del ámbito de la interfaz `TextualConventions` para convertir el valor tipado a un `string` y viceversa.
4. Traducir el valor de la invocación de las macros `MODULE-IDENTITY` en una constante literal IDL de tipo `string`.
5. Traducir el valor de la invocación de las macros `OBJECT-IDENTITY` en una constante literal IDL de tipo `string`.
6. Cada grupo se traduce en una interfaz IDL (G en la figura 16), y una interfaz IDL (T en la figura 16) para cada entrada de cada una de las tablas del grupo:
   - Las variables no tabuladas de un grupo se traducen como atributos (`atribute`) dentro del ámbito de la interfaz IDL del grupo. Las variables de las columnas de las tablas se traducen como atributos dentro del ámbito de la interfaz IDL de la entrada de la tabla:
   - Usar el descriptor de la macro `OBJECT-TYPE` de la variable correspondiente como identificador del `atribute`.
   - Obtener el tipo IDL del atributo de cláusula `SYNTAX` de la macro `OBJECT-TYPE` de la variable correspondiente.
   - Obtener el modo del atributo de la cláusula `MAX-ACCESS` de la macro `OBJECT-TYPE` de la variable correspondiente.

- Si hay una cláusula `DEFVAL`, definir una operación dentro del ámbito de la interfaz `DefaultValues` que devuelve el valor por defecto de forma tipada.

7. Traducir el valor de la invocación de las macros `NOTIFICATION-TYPE` en una constante literal IDL de tipo `ASN1_ObjectIdentifier`:
   - Traducir la especificación de valor de la cláusula `OBJECTS` a una estructura IDL que tiene un *item* por cada variable en la cláusula `OBJECT`; el tipo de *items* de la estructura se definen como el trío nombre-índice-valor y el tipo del valor se deriva de la variable que está siendo traducida.
   - Definir una operación en el ámbito de la interfaz `SnmpNotifications` de este módulo para comunicación de sucesos de tipo *push*. Los parámetros de entrada (`in`) de la operación son: el OID de la parte SNMP fuente, el OID del contexto, un *time-stamp* del suceso y la estructura IDL definida por la especificación de valores de la cláusula `OBJECTS`. Usar el descriptor de la macro como el identificador de la operación.
   - Definir dos operaciones (`try_<op>` y `pull_<op>`) dentro del ámbito de la interfaz `PullSnmpNotifications` de este módulo, donde `<op>` es el identificador de la correspondiente operación en la interfaz `SnmpNotifications`. Los parámetros de las operaciones son los mismos que en el modelo de *push* de eventos, pero como parámetros de salida (`out`).
8. Asignar los OIDs de las macros como `RepositoryId` (usando `#pragma ID declaración`) a todos los identificadores IDL que corresponden a un descriptor de macro.
9. Ignorar las macros relativas a las reglas `MODULE-COMPLIANCE.`

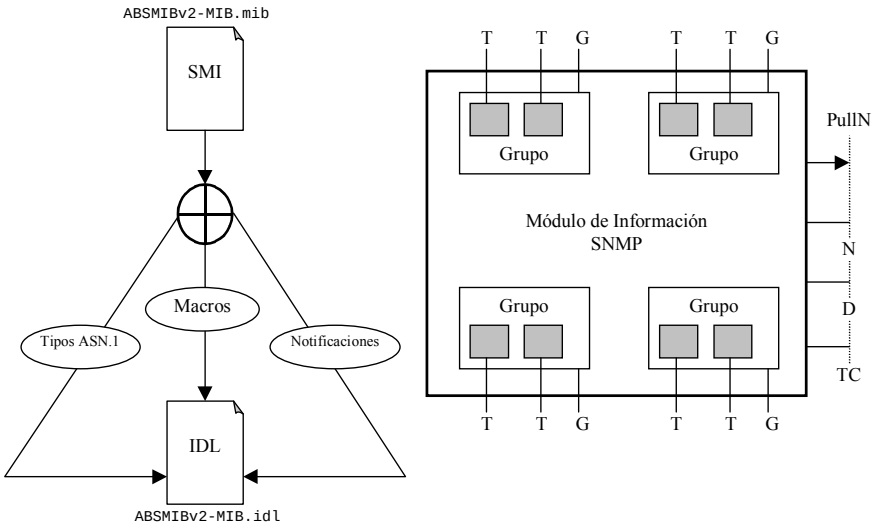


Figura 16. ***Traducción de SMI a IDL***

Posteriormente se ha hecho modificaciones a la Traducción de especificaciones referentes a SMI [OMG98c]. Principalmente, en lo que se refiere a los ficheros idl estándar, pero también en lo que se refiera a las tablas. Se incluyen a continuación:

1. Las interfaces `CosLifeCycle::LifeCycleObject` y `CosPropertyService::CosPropertySet` se definen como base para `SNMPMgmt::SmiEntry`;
2. La macro `DESCRIPTION` ahora se traduce con "`/**`" al comienzo del comentario
3. La macro `OBJECT-TYPE` para las Tablas se pone como un atributo IDL dentro del ámbito de la interfaz IDL del grupo al que pertenece (en vez de una cadena constante)
4. La cláusula `INDEX` para la macro `OBJECT-TYPE` en tablas ahora se traduce como una constante IDL de tipo `string`.
5. La especificación de valor en la cláusula `AUGMENT` de la macro `OBJECT-TYPE` en tablas se usa ahora como tipo base en vez de `SNMPMgmt::SmiEntry`.
6. Se ha definido una interfaz genérica de fábrica en el módulo `SNMPMgmt` y un esquema de traducción para generar interfaces de fábrica para entradas de grupos de tablas específicos de la MIB
7. Las interfaces relativas a notificaciones se han pasado del módulo `SNMPMgmt` a otro, llamado `SNMPv2`, en otro fichero, `SNMPv2.idl`; los nombres de las interfaces IDL han pasado a `Notifications` y `PullNotifications`.

8. El módulo de *Traps* de SNMPv1 se llama ahora `SNMPv1`, y el fichero `SNMPv1.idl`. Los nombres de las interfaces IDL también se han cambiado a `Notifications` y `PullNotifications`.
9. Para los módulos de MIB de SNMPv2, las interfaces específicas de la MIB, `Notifications` y `PullNotifications` tienen sus interfaces base `SNMPv2::Notifications` y `SNMPv2::PullNotifications`, respectivamente. Para los de SNMPv1, sus interfaces son `SNMPv1::Notifications` y `SNMPv1::PullNotifications`, respectivamente.

Los `#pragma ID` se consideran obsoletos. El compilador deberá generar un fichero que relacione los nombres IDL, con los OID. Este fichero es necesario en la pasarela entre CORBA y SNMP.

En el siguiente cuadro se muestra un ejemplo de lo que puede suponer esta traducción. Para ello se ha realizado sobre el cuadro de ejemplo de SMI descrito en el capítulo de Gestión de Red (cuadro 1) los algoritmos de traducción de especificaciones de JIDM.

```
Module ABSMIBv2_MIB {
    Interface bsm : SNMPMgmt::SmiEntry {
        /**
  DESCRIPTION:"The name of the file where to log the
  notifications"
  */
  attribute DisplayStringType bsmLogFileName;
    };
    /**
DESCRIPTION:"Information about every notifications in the last
session"
*/
const string bsmNotificationTable=
   "::ABSMIBv2_MIB::bsmNotificationTable";
    /**
DESCRIPTION:"Sequence of information about any notification"
*/
interface bsmNotificationTableEntry : SNMPMgmt::SmiEntry {
          /**
    INDEX : { bsmNotificationIndex}
    */
          const string EntryIndexVarList = "bsmNotificationIndex";
          /**
```

```
        DESCRIPTION:"Index of BsmNotificationTable"
        */
              readonly attribute ASN1_Integer bsmNotificationIndex;
              /**
        DESCRIPTION:"Time from the launch when the notification was
        received"
        */
              readonly attribute TimeTicksType bsmNotificationTime;
              /**
        DESCRIPTION:"Name of the notification received"
        */
              readonly attribute DisplayStringType bsmNotificationType;
          };
      };
```

Cuadro 2. ***Ejemplo de traducción entre SMI e IDL***

1.3.2.2. Traducción de Interacciones (JIDM-IT)

La Traducción de Interacciones cubre el proceso por el cual las interacciones de un dominio se traducen en una o más interacciones en el otro dominio. Una pasarela (*gateway*), la entidad responsable de traducir las interacciones, puede recibir una PDU de CMIP, traduciendo entonces ésta en una o más peticiones o respuestas de interfaces IDL. Por ejemplo, si se recibe una petición GET de CMIP con *scoping* (ámbito) y filtrado, la pasarela tendrá que identificar el conjunto de objetos que concuerdan con el filtro dentro del ámbito pedido e invocar las operaciones apropiadas en cada uno de esos objetos. Los resultados deben ser recogidos y formateados en una o más PDUs CMIP de respuesta. Es responsabilidad de la especificación de la Traducción de Interacciones definir como se realiza este tipo de operaciones.

Esto es claramente una parte fundamental del trabajo de JIDM. Hasta la fecha ha habido grandes esfuerzos para abordar el tema. Con ello, la producción de una pasarela para un conjunto particular de definiciones GDMO o SMI es totalmente factible. La cuestión principal y más destacable es proporcionar acceso dinámico en el dominio CORBA para la información de gestión de dichas especificaciones; por ejemplo, valores por defecto, rango de valores no representables en IDL, comportamientos, etc. Este acceso dinámico permite la construcción de herramientas genéricas más potentes y, en particular, que se pueda construir una pasarela que sea independiente de un modelo de objetos en particular.

En suma, la Traducción de Interacciones debe cubrir la inicialización, identificando cómo se inicializa y utiliza la pasarela, como identifica la población de objetos existentes y que otros ejemplares de servicios puede necesitar. La pasarela necesitará probablemente interactuar con servicios estándar del dominio CORBA, como por ejemplo, el servicio de nombres para resolver nombres unívocos (*Distinguished Names*), el servicio de ciclo de vida para crear nuevos ejemplares de objetos y canales de eventos para la distribución de los mismos.

La Traducción de Interacciones requiere la captura de las interacciones por una pasarela que las convierta de acuerdo con las reglas de traducción. Entonces, en los escenarios OSI/CORBA, la pasarela deberá:

1. Recibir cualquier petición CMIP de SET, GET o ACTION y traducirla en una o más invocaciones a métodos soportados por el (los) objeto(s) destino.
2. Recibir cualquier evento generado por un objeto-aplicación y traducirlo en una petición REPORT que será reenviada a sistemas remotos que hubieran mostrado su interés en recibir eventos.
3. Recibir la invocación de métodos y reenviarlos como peticiones SET/GET/ACTION de CMIP a algún agente OSI.
4. Recibir peticiones EVENT-REPORT de CMIP y reenviarlas como eventos CORBA a las partes interesadas en el dominio CORBA.
5. Recibir cualquier petición CREATE/DELETE de CMIP y traducirla en una invocación a un método que soporte un objeto fábrica (*factory object*).
6. Recibir invocación de métodos para la creación y borrado de objetos en un sistema remoto y reenviarlos a algún agente OSI como peticiones CREATE/DELETE de CMIP.

Esta conversión de protocolos es complicada en lo que se refiere a la necesidad de traducir identificadores debido a las diferencias entre los ámbitos y la sensibilidad a las mayúsculas de GDMO e IDL, para traducir nombres unívocos de GDMO y referencias de objetos de CORBA, y para manejar las peticiones con *scoping* y filtrado que puede requerir una petición CMIP para traducirse en múltiples secuencias de operaciones IDL.

Aunque puede parecer deseable traducir el tipo `ObjectInstance` de CMIP, directamente a referencias de objeto CORBA, esta no es la traducción correcta debido a la diferencia semántica. La traducción pasa el tipo `ObjectInstance` exactamente como se haría con cualquier otro tipo compuesto ASN.1, es decir, manteniendo el formato de nombre unívoco. El uso del nombre unívoco es consistente en sí mismo y no requiere el proceso de Traducción de Especificaciones para traducir `ObjectInstance` como un caso especial. El

proceso de Traducción de Interacciones también requiere la capacidad de convertir entre `ObjectInstance` y referencias de objetos de CORBA, por lo que existirán las funcionalidades en tiempo de ejecución para realizar la traducción para una aplicación en caso de ser necesario.

En el caso de SNMP se presentan casos parecidos. A continuación se dedica un apartado completo a la traducción de interacciones entre CORBA y SNMP.

1.3.3. Traducción entre CORBA y SNMP

El presente apartado pretende dar una visión de cómo crear una pasarela en un escenario CORBA/SNMP. Se basa los servicios SNMP definidos por la Traducción de Interacciones de JIDM [OMG98c], al estilo de los COSS (*CORBA Object Standard Services*, Servicios Estándar de Objetos CORBA). De ellos, cabe destacar el servicio de nombres SNMP, que será el que se utilice a la hora de localizar los objetos, para poder hacer *get* y *getnext*.

La meta principal al traducir nombres del dominio de SNMP al de CORBA es normalizar la jerarquía de nombres de SNMP (*host*, variable, índice) basada en la interfaz `NamingContext` del servicio de nombres de CORBA. Esta meta se consigue de dos formas: Estandarizando la jerarquía del árbol de la MIB y extendiendo la interfaz `NamingContext` para listar sus entradas en orden lexicográfico.

Cuando las interfaces basadas en IDL para entradas de una tabla o grupos se implementan en el dominio de CORBA, la implementación debe soportar la cláusula `INDEX` en que se basa su nombramiento. El nombramiento de entradas de la MIB debe hacerse de tal forma que alguien pueda usar el servicio de nombres de CORBA para obtener las referencias a los objetos de las entradas de la MIB usando sus nombres SNMP. Los nombres de las entradas de la MIB también son registrados con el servicio de nombres de tal forma que se puede obtener las entradas en el orden lexicográfico de un `GET-NEXT` sin mayor sobrecarga. Por ello se define una nueva interfaz, llamada `SnmpNaming::NamingContext`, que extiende el módulo `CosNaming` para soportar la obtención de entradas de la MIB usando nombres SNMP.

La Figura 17 describe la arquitectura del servicio de nombres SNMP y la jerarquía de herencia de `SnmpNaming::NamingContext`. La interfaz `NameDirectory` también actúa como interfaz de administración para la gestión del espacio de nombres SNMP dentro del dominio CORBA.

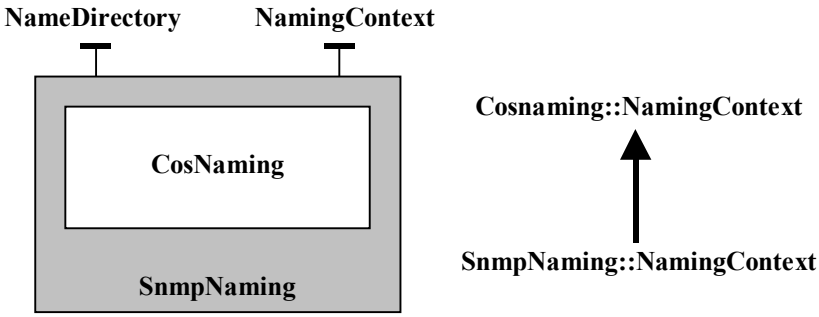


Figura 17. ***La arquitectura para el Servicio de Nombres SNMP y la jerarquía de herencia de interfaces***

1.3.3.1. Visión general del Servicio de Nombres de CORBA.

La especificación del Servicio de Nombres de CORBA define como asociar un nombre a un objeto (llamado *name binding) (aplicación* o asociación de nombres) y se asemeja al nombrado de ficheros y directorios en el sistema de ficheros de UNIX. Un *name binding* se define siempre relativo a un contexto de nombres (*naming context*). Un contexto de nombre es un objeto que contiene un conjunto de *name bindings* en el que cada nombre es único. El contexto de nombres puede ser aplicado a un nombre. Se pueden aplicar nombres diferentes a un mismo objeto en un mismo o distintos contextos al mismo tiempo. Resolver un nombre es determinar los objetos asociados con el nombre en un contexto dado. Un nombre está ordenado por una secuencia de componentes de nombre. Un componente de nombre consiste en dos elementos: `id` y `kind`, como se puede ver en cuadro 3. Un nombre con múltiples componentes se llama nombre compuesto. El cuadro 3 describe las interfaces y tipos de datos necesarios para soportar el servicio de nombres de CORBA. La interfaz `NamingContext` soporta las operaciones necesarias para crear un nuevo contexto de nombres, asociar un nombre a un objeto, desasociarlo, resolver un nombre y listar todos los nombres asociados a un contexto de nombres. En los siguientes puntos sólo se usará el atributo `id`, olvidando el `kind` (inicializándolo a una cadena de longitud nula).

```
Module CosNaming {
   typedef string Istring;
   struct NameComponent {Istring id; Istring kind;};
   typedef sequence<NameComponent Name;
   ...
   interface NamingContext {...};
};
```

Cuadro 3. ***Tipos de datos e interfaces de la especificación del Servicio de Nombres de CORBA***

1.3.3.2. Uso de nombres SNMP en el dominio CORBA.

En este punto se describirá como nombrar entradas de la MIB SNMP en el dominio CORBA usando las variables `INDEX` y acceder a las entradas de la MIB basándose en los nombres SNMP correspondientes y finalmente, obtener los valores de las variables SNMP.

En el dominio de SNMP los nombres están asociados con la ejemplarización de variables y los ejemplares de variables, tabuladas o no, están direccionadas en el ámbito de una dirección IP de un *host* o máquina anfitriona. La noción de entradas de una tabla se conceptualiza mediante la asignación del mismo índice a todos los ejemplares de variables de la misma entrada o fila. Desde la perspectiva del gestor, las entradas de las MIBs SNMP están tratadas implícitamente con la siguiente jerarquía: *Host*, OID de la variable e índice de fila. Dado que las MIBs SNMP son dependientes de la localización del agente, los nombres SNMP son así mismo dependientes de la localización. Para las variables no tabuladas de un grupo, el nombre es siempre cero ("`0`") y para filas de variables tabuladas, el ejemplar de información depende de la cláusula `INDEX` de la tabla correspondiente.

Para traducir los nombres de ejemplares de variables en el dominio de SNMP a atributos de entradas de MIB en el dominio de CORBA, se necesita traducir los nombres de variables dentro del ámbito de un *host* a un árbol jerárquico de nombres basado en la especificación del servicio de nombres de CORBA. La Figura 18 ilustra una posible implementación del árbol de nombres.

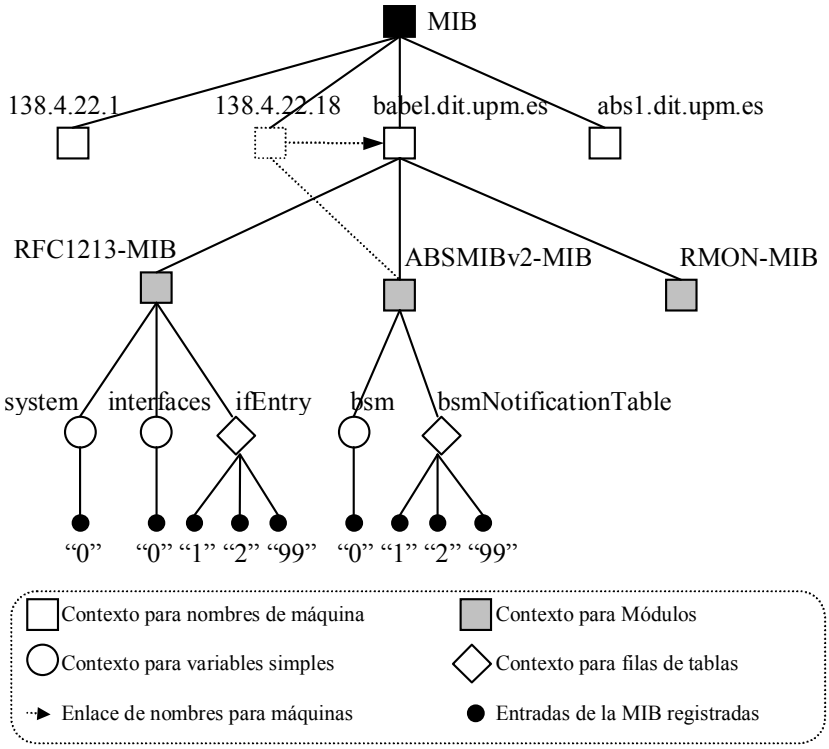


Figura 18. ***La jerarquía basada en el contexto de nombres para el árbol de nombres de la MIB SNMP.***

La jerarquía básica de los nodos del árbol de nombres es como sigue: MIB (raíz global de todas las MIBs SNMP), nombre de *host*, nombre del módulo de información de la MIB, nombres de las interfaces IDL para las entradas de tabla o grupos, y finalmente, los índices de las filas, que son las hojas del árbol. Los nodos raíz, *host*, módulo y tabla se soportan por la interfaz `SnmpNaming::NamingContext` y las referencias a las entradas de la MIB se registran con índices filas con los nodos para tablas/grupos.

La raíz de la MIB global se registra como un nombre bien definido, `MIB`, dentro del ámbito de algún contexto de nombres en el espacio de nombres de CORBA. La función `resolve_mib_root()` de la interfaz `NameDirectory` se puede usar para encontrar la referencia de la raíz de la MIB. La `MIB` raíz global es un contexto de nombres para todos los *hosts* que implementan la MIB SNMP.

Los nodos para las MIBs SNMP de los *hosts* se representan por un contexto de nombres (dentro del ámbito del nodo raíz MIB). Dado que un nombre de host puede representarse de muchas maneras (dirección IP, nombre DNS, cualquier alias que posea...), se necesita definir un nombre que se pueda usar sin ambigüedades identificando los nodos de los

hosts bajo MIB. El esquema por defecto para nombrar un nodo para un *host* es el nombre basado en DNS (por ejemplo: `babel.dit.upm.es`). Si no se soporta el nombre basado en DNS, entonces se usará la dirección IP (por ejemplo: `130.4.22.18`). Puesto que el servicio de nombre de CORBA permite alias de nombres (como enlazar ficheros en UNIX), una implementación podría soportar tanto el nombrado basado en DNS como el basado en la dirección IP. Dado que los componentes de nombre CORBA poseen dos partes: `id` y `kind`, se traduce el nombre SNMP como cadena de caracteres a la parte `NameComponent.id` del nombre CORBA y se ignora la parte `NameComponent.kind`, inicializándola con una cadena de longitud nula. Una implementación del árbol de nombres puede añadir más nodos entre la raíz y los *host* basada en la jerarquía de dominios DNS o el *subnetting* IP, siempre que la resolución de nombres basada en nombres completos de DNS o dirección IP devuelva la referencia al mismo nodo.

Dentro del ámbito de cada *host*, habrá un contexto de nombres para cada uno de los módulos de información de la MIB que están implementados en el *host*. El contexto de nombres asociado con el nodo del módulo de la MIB se nombra el identificador de módulo en IDL.

En el ámbito de cada módulo de información, habrá un nodo basado en contexto de nombres para cada uno de las interfaces IDL para grupos y tablas definidos dentro del ámbito del módulo IDL. Los contextos de nombres asociado con los nombres para grupos y tablas se nombran usando los identificadores de las correspondientes interfaces IDL.

Las entradas (ejemplares de objetos que soportan las interfaces IDL obtenidas por la traducción de la MIB SNMP) de la implementación de la MIB se asocian dentro del contexto de nombres de la correspondiente interfaz IDL. Los nombres de las entradas de la MIB asociadas son la información del ejemplar necesaria para identificar todas las variables de la fila de la tabla conceptual. Para variables no tabulares del grupo, el nombre es siempre cero ("`0`") y para filas de una variable tabulada, la información del ejemplar depende de la cláusula `INDEX` de la tabla correspondiente. Los objetos asociados son las hojas del árbol de nombres.

```
MIB--babel.dit.upm.es-ABSMIBv1-MIB-bsmConfEntry-"1"
raíz-host------------módulo-------tabla--------referencia_a_la_fila
MIB--babel.dit.upm.es-ABSMIBv1-MIB-bsm---"0"
raíz-host------------módulo-------grupo-referencia_al_objeto
```

Cuadro 4. ***Caso particular de ABS***

Como se puede ver en cuadro 4, en el caso particular que atañe, en cualquier caso se obtendría la referencia al objeto que está manteniendo las variables. En el caso de una tabla, hay un objeto fila servidor de las variables de su fila, y en el caso de variables simples, el objeto servidor mantiene todas las variables simples.

1.3.3.3. Traducción de nombres

El cuadro 5 describe como traducir el nombre SNMP de un ejemplar de una variable no tabulada de un grupo a un nombre de la correspondiente entrada de la MIB y atributo IDL en el dominio CORBA. Dado un nombre no tabulado en su forma OID, primero hay que separar la información del ejemplar del OID de la variable usando el prefijo más largo del OID de la variable que case, como se puede ver en la segunda línea. Después, se deriva el OID del grupo quitando el identificador de más a la derecha de la variable (como se ve en la tercera línea). Los OID de grupo y de variable se convierten entonces a los correspondientes nombres *scoped* IDL (usando la interfaz `SMI::Repository`). Después, se separa el nombre IDL para el grupo en su módulo y nombre de interfaz. Finalmente, se usa la secuencia ordenada de nombre de módulo, interfaz e información del ejemplar para derivar un nombre compuesto de la entrada de la MIB para el grupo dentro del contexto de un *host*. Si se sabe el nombre del *host* en dirección DNS o IP, se sabe la ruta completa del nombre del grupo dentro del ámbito del nodo raíz, llamado MIB.

```
    evalSlot.0 (->1.3.6.1.3.555.2.1.0)
-> 1.3.6.1.3.555.2.1,   0
-> 1.3.6.1.3.555.2, 0, 1.3.6.1.3.555.2.1
-> FIZ_MIB::eval,   0, FIZ_MIB::eval::evalSlot
-> FIZ_MIB, eval,   0, FIZ_MIB::eval::evalSlot
-> <FIZ_MIB, eval, 0, evalSlot>
-> host_nc-resolve(<FIZ_MIB, eval, 0>)->get_property("evalSlot")
      -> DII-GET(host_nc-resolve(<FIZ_MIB, eval, 0>), "evalSlot"))
```

Cuadro 5. ***Traducción de variables no tabuladas a nombres CORBA y atributos IDL***

En el cuadro 5 se ve como acceder al valor de un ejemplar de `evalSlot.0` (el cual se representa realmente por `1.3.6.1.3.555.2.1.0`) al nombre IDL de la correspondiente interfaz de grupo (`FIZ_MIB::eval`) y atributo (`FIZ_MIB::eval::evalSlot`) del ejemplar. El nombre compuesto del objeto (dentro del ámbito de un *host*) que soporta `eval` se deriva traduciendo el nombre completo de la interfaz y la información de ejemplar (`"0"`) a la parte

del componente `id` del nombre compuesto. Dado el nombre compuesto, se puede obtener la referencia a la entrada de la MIB que representa el grupo `eval` usando la operación `resolve()` del contexto del nombre del *host*. Entonces se puede usar la referencia a objeto hallada para solicitar el valor de la variable invocando la operación de `get` asociada con el atributo `evalSlot`. El valor de la variable se puede obtener invocando la operación de `get` directamente al atributo usando DII (Interfaz de Invocación Dinámica) o se puede obtener usando la operación `PropertySet::get_property()` estrechando (*narrowing*) la referencia resuelta a un tipo de interfaz `SmiEntry`.

El cuadro 6 describe como traducir el nombre SNMP de un ejemplar de variable tabulada de una tabla conceptual a nombres basados en el servicio de nombres de una entrada de la MIB y el atributo de la correspondiente interfaz. En dicha tabla, primero se separa el identificador de objeto (`1.3.6.1.3.555.2.2.1.4.2`) del OID de la variable (`1.3.6.1.3.555.2.2.1.4`) y su información de ejemplar (`2`) usando el casado de prefijo más grande. Entonces, se deriva el OID (`1.3.6.1.3.555.2.2.1`) de la correspondiente entrada de tabla quitando el último número del identificador de variable. Luego se obtienen los OIDs de la entrada de la tabla y la variable al correspondiente nombre IDL del interfaz (`FIZ_MIB::evalEntry`) y atributo (`FIZ_MIB::evalEntry::evalStatus`), respectivamente. Entonces se separa el nombre IDL de la entrada de tabla en nombre de módulo y de interfaz. Luego se usa la secuencia ordenada del nombre de módulo, interfaz y la información del ejemplar para derivar el nombre compuesto (`<FIZ_MIB, evalEntry, 2, evalStatus>`), dentro del ámbito del contexto del *host*. Este nombre compuesto representa el nombre de la entrada de la MIB que representa la fila dentro de la tabla. Dado el nombre compuesto, se obtiene la referencia a la entrada de la MIB usando la operación `resolve()` del contexto del nombre del *host*. Entonces se puede usar la referencia a objeto hallada para solicitar el valor de la variable invocando la operación de `get` asociada con el atributo `evalStatus`. El valor de la variable se puede obtener invocando la operación de `get` directamente al atributo usando DII (Interfaz de Invocación Dinámica) o se puede obtener usando la operación `PropertySet::get_property()` estrechando (*narrowing*) la referencia resuelta a un tipo de interfaz `SmiEntry`.

```
    evalStatus.2 (->1.3.6.1.3.555.2.2.1.4.2)
  -> 1.3.6.1.3.555.2.2.1.4, 2
  -> 1.3.6.1.3.555.2.2.1,   2, 1.3.6.1.3.555.2.2.1.4
  -> FIZ_MIB::evalEntry,    2, FIZ_MIB::evalEntry::evalStatus
  -> <FIZ_MIB, evalEntry, 2,  evalStatus>
```

```
   -> host_nc-resolve(<FIZ_MIB, evalEntry, 2>)->get_property("evalStatus")
         -> DII-GET(host_nc-resolve(<FIZ_MIB, evalEntry, 2>),
"evalStatus"))
```

Cuadro 6. ***Traducción de nombres de variables tabuladas a nombres CORBA y atributos IDL.***

1.3.3.4. Resolución de nombres

El cuadro 7 describe en pseudo código como resolver el nombre de un objeto (como se describía anteriormente) a la correspondiente referencia a objeto. Primero se necesita encontrar la referencia al contexto de nombres para la raíz del árbol de nombres para la MIB SNMP. Entonces, se encuentra la referencia al contexto de nombres para el *host*. Luego, se toma la referencia del Repositorio SMI y se usa esta referencia para separar el identificador de objeto de la variable en un nombre IDL (de la forma `::M::I::V`) y la parte del índice.

Entonces se forma un nombre compuesto (del tipo `CosNaming::Name`) donde sus componentes son (en orden): Nombre del módulo (M), nombre del tipo de interfaz (I) y la información del índice de la variable. Entonces, se invoca la operación `resolve()` del contexto del nombre del *host* usando el nombre compuesto para obtener la referencia asociada con la entrada de la MIB.

Una vez se obtiene la referencia a la entrada de la MIB, el valor de la variable se puede obtener de dos formas: usando la interfaz `PropertySet` o la invocación dinámica. La operación `get_property("evalStatus")` de la interfaz `PropertySet` se invoca usando el nombre de la variable para obtener el valor de la variable. Dado que las variables se traducen como atributos IDL, se puede usar la invocación dinámica (DII) para obtener el valor de la variable como cualquier atributo IDL.

```
CosNaming::NamingContextRef snmp_mib_root;
CosNaming::NamingContextRef host_naming_context;
CosNaming::Name * cos_name;
CosNaming::NameComponent name_comp[3];
CORBA::ObjectRef obj;
         SnmpNaming::NameDirectory snmp_name_dir;
   // encuentra la referencia usando el servicio de búsqueda
   snmp_mib_root = snmp_name_dir-get_root_mib();
   // obtiene get the naming-context using the host
   name_comp[0].id("babel"); name_comp[0].kind("");
   cos_name = new CosNaming::Name(1,1,&name_comp);
   obj = snmp_name_dir-resolve(*cos_name);
```

```
   host_naming_context = CosNaming::NamingContext::_narrow();
      SNMPMgmt::SmiEntryRef smi_entry;
CORBA::Any * anyValue;
         ASN1_ObjectIdentifier varObjId;
   // convierte esto a CosNaming::Name y var_name
         SMI::RepositoryRef smi_repo;
   // encuentra la referencia usando el servicio de búsqueda
         ASN1_ObjectIdentifier varOID, varIndex;
   smi_repo-split_var_object_id(varObjId, &varOID, &varIndex);
         SMI::ScopedName varScopedName; // de la forma ::M::I::V
   varScopedName = smi_repo-get_scoped_name(varOID);
         //Separa el varScopedName en módulo, interfaz y nombre de
variable
   CORBA::String varName, varEntryName, varModuleName, tempPtr;
   tempPtr = strtok (varScopedName, ":");
   for (CORBA::Long i =0; tempPtr!=NULL; i++) {
      switch (i) {
         case(0): varModuleName =CORBA::string_copy(tempPtr); break;
         case(1): varEntryName =CORBA::string_copy(tempPtr); break;
         case(2): varName =CORBA::string_copy(tempPtr); break;
      }
      tempPtr = strtok(NULL, ":");
   }
   // obtiene la referencia del objeto para evaluar el grupo usando el
   // tipo de interfaz e índice
   name_comp[0].id(varModuleName); name_comp[0].kind("");
   name_comp[1].id(varEntryName); name_comp[1].kind("");
   name_comp[2].id(varName); name_comp[2].kind("");
   cos_name = new CosNaming::Name(3, 3, &name_comp);
   obj=host_naming_context-resolve(*cos_name);
   // obtiene el valor del atributo usando la interfaz SmiEntry
   smi_entry = SNMPMgmt::SmiEntry::_narrow(obj);
   anyValue = smi_entry-get_property(VarName);
```

Cuadro 7. ***Acceso al valor de un atributo de un objeto usando el nombre SNMP***

Aunque las referencias a las entradas de la MIB se obtienen usando nombres SNMP dependientes de la localización (debido al nombre de *host*), una vez se obtiene la referencia a un objeto, un programador de aplicaciones no tiene que preocuparse de la información de índice. El uso de las referencias a objeto a entradas de la MIB es independiente de la localización, como cualquier otra referencia CORBA.

El cuadro 8 describe en pseudo código la traducción del `getnext` sobre una variable a operaciones de servicios SNMP de interfaces. Se ha usado el `SMI::Repository` para obtener información sobre el nombre IDL de la variable, el OID de la siguiente variable y `SnmpNaming::NamingContext` para acceder a las entradas de la MIB en orden lexicográfico. Dado que la operación `CosNaming::NamingContext::list()` puede no dar un orden lexicográfico de los nombres de las entradas de la MIB, sin las extensiones `SnmpNaming`, no se podrá obtener las entradas de la MIB en orden lexicográfico.

```
GET-NEXT (1.3.6.1.3.555.2.2.1.3.2) // GET-NEXT(evalValue.2)
        // Inicializa varObjId al OID de la variable
   ASN1_ObjectIdentifier varObjId = "1.3.6.1.3.555.2.2.1.3.2";
   ASN1_ObjectIdentifier nextVarObjId, nextVarOID, nextVarIndex;
   // a devolver
   CORBA::Any anyValue; // a devolver
         ASN1_ObjectIdentifier varOID, varIndex;
   smi_repo-split_var_object_id(varObjId, &varOID, &varIndex);
   // En este punto varOID == 1.3.6.1.3.555.2.2.1.3 y varIndex == 2
         SMI::ScopedName varScopedName; // de la forma ::M::I::V
   varScopedName = smi_repo-get_scoped_name(varOID);
   // varScopedName es ahora FIZ_MIB::evalEntry::evalValue
   // separa varScopedName en módulo, interfaz y nombre de variable
   CORBA::String varName, varEntryName, varModuleName, tempPtr;
   tempPtr = strtok (varScopedName, ":");
   for (CORBA::Long i =0; tempPtr!=NULL; i++) {
      switch (i) {
         case(0): varModuleName =CORBA::string_copy(tempPtr); break;
         case(1): varEntryName =CORBA::string_copy(tempPtr); break;
         case(2): varName =CORBA::string_copy(tempPtr); break;
      }
      tempPtr = strtok(NULL, ":");
   }
   // obtiene la referencia del objeto del NamingContext asociado con
   // la entrada de la tabla
   CORBA::ObjectRef obj;
   name_comp[0].id(varModuleName); name_comp[0].kind("");
   name_comp[1].id(varEntryName); name_comp[1].kind("");
   name_comp[2].id(varName); name_comp[2].kind("");
   cos_name = new CosNaming::Name(3, 3, &name_comp);
   obj=host_naming_context-resolve(*cos_name);
   entry_snmp_nc = SnmpNaming::NamingContext::_narrow(obj);
```

```
SnmpNaming::NamingIteratorRef name_iter;
name_iter = entry_snmp_nc-get_nc_entry_iterator(varIndex);
SNMPMgmt::SmiEntry smi_entry;
if (name_iter-next_one_entry(&obj) == TRUE) {
   ASN1_ObjectIdentifier nextVarOID, nextVarIndex;
   smi_entry = SNMPMgmt::_narrow(obj);
   nextVarOID = varOID;
   nextVarIndex = smi_entry-name();
   nextVarObjId = nextVarOID + nextVarIndex;
   anyValue = smi_entry-get_property(varName);
   return (nextVarObjId, anyValue);
}
// varIndex se refiere a la última entrada - ir a la primera entrada
// de la siguiente variable
next_oid = smi_repo-get_next_oid(1.3.6.1.3.555.2.2.1.3);
// si no hay más OID, devolver la excepción EndOfMib
return GET-NEXT(next_oid);
// en la siguiente iteración, la varIndex es una cadena nula.
```

Cuadro 8. ***Acceso al valor de un atributo de una variable con GET-NEXT***

1.3.3.5. Gestión de objetos CORBA usando SNMP

Este punto describe como hacer uso de las reglas de JIDM, traduciendo una especificación SMI a IDL, para administrar objetos CORBA nativos mediante el protocolo SNMP.

La aproximación se basa en el marco de objetos con múltiples interfaces definidas en el modelo computacional de TINA-C donde los objetos gestionados soportan múltiples tipos de interfaces. Esto no concuerda exactamente con el modelo de OMA, pero el problema es únicamente conceptual. Alguna de las interfaces que soporta el objeto define la información específica de gestión que se puede obtener de la especificación de la MIB.

En esta aproximación, la especificación de la MIB basada en SMI para grupos, tablas y variables se define primero. Se usa la especificación de la MIB SNMP para el objeto CORBA como su especificación de instrumentación que proporciona únicamente la mínima información necesitada para la gestión. La MIB SNMP también proporciona un esquema de nombres para los objetos (basado en variables en la cláusula `INDEX`) que no está disponible en el dominio CORBA. (No hay forma de poder especificar como nombrar un objeto en el dominio CORBA. Esto es una característica especial de las especificaciones de gestión de red.)

Una vez se define la especificación de la MIB SNMP, se usa la aproximación de la Traducción de Especificaciones de JIDM para generar las interfaces IDL basándose en la especificación de la MIB SNMP. A continuación se podrían extender las interfaces IDL generados de la MIB SNMP para añadir capacidades mejoradas de gestión para gestores basados en CORBA. Por ejemplo, si se tiene una tabla SNMP `mibTableEntry` en el módulo `Example-MIB`, esta se traducirá en una interfaz IDL `mibTableEntry` como se muestra en el cuadro 9. Se puede entonces extender la interfaz generada y definir una nueva, que se llame `MgmtInterface`, como se ve en el cuadro 10, de forma que se pueda añadir operaciones y atributos *ad oc* basados en tipos IDL arbitrarios

```
Module Example_MIB {
   interface mibTableEntry : SNMPMgmt::SmiEntry {
      <Todas las variables columna irían aquí como atributos>
   };
};
```

Cuadro 9. ***Descripción en IDL del módulo Example_MIB con mibTableEntry***

```
Module Extended_Example_MIB {
   interface MgmtInterface : Example_MIB::mibTableEntry {
     <Se añade atributos y operaciones de gestión específicas de CORBA>
   };
};
```

Cuadro 10. ***Extensión de la interfaz de gestión basada en SNMP***

En esta aproximación no sólo se proporciona la capacidad de gestión basada en SNMP, sino que también se extienden las interfaces IDL basadas en SNMP para añadir cualquier capacidad que fuera necesaria para gestores basados en CORBA Hay que reseñar que se puede definir tanto atributos como operaciones basados en tipos de datos complejos de IDL en la interfaz IDL extendida. No es necesario limitarse a los tipos enteros o cadenas de caracteres permitidos en SNMP.

Durante la implementación de `MgmtInterface` se puede implementar los métodos *get* y *set* de los atributos de la interfaz `mibTableEntry` como instrumentación relacionada con las variables SNMP, así como los atributos y operaciones definidos en el `MgmtInterface`. En suma, la referencia del objeto asociado con el ejemplar del objeto que soporta `MgmtInterface` se tiene que registrar en el servicio de nombres de CORBA usando el nombre SNMP. Los nombres SNMP se derivarán basándose en los valores de las variables `INDEX` de la `mibTableEntry` y siguiendo la política de nombrado de SNMP.

Esta aproximación sólo se puede aplicar si se está interesado en soportar la gestión de objetos CORBA desde un gestor basado en SNMP así como basados en CORBA y la implementación del objeto CORBA está todavía por hacer.

1.3.4. Conclusiones

El presente capítulo ha tratado de dar una visión de la gestión de aplicaciones distribuidas. Tras una introducción que incluye las distintas aproximaciones publicadas, ha desarrollado la aproximación dada por JIDM, en que se posibilita la interacción entre CORBA y los dos estándares de gestión que actualmente están en vigencia: OSI-SM y SNMP. El uso e interacción de CORBA y SNMP para este proyecto se ha visto posibilitado en gran medida por las Traducciones de Especificaciones e Interacciones que propone JIDM.

En el siguiente capítulo se presenta un ejemplo de aplicación distribuida. Sobre ella se aplicarán los conceptos aprehendidos previamente en estos tres capítulos para desarrollar el sistema de gestión que administrará dicha aplicación distribuida.

1.4. ABS, Arquitectura para un Servicio de Intermediación de Información

1.4.1. Introducción

El rápido desarrollo y disponibilidad de aplicaciones de comercio electrónico y servicios de información sobre las infraestructuras de comunicación existentes ha llevado a la necesidad de unos nuevos servicios, denominados servicios de intermediación electrónica (más conocidos por su nombre original en inglés, *electronic brokerage*), que tratan de acercar los dominios de los clientes y los suministradores en el ámbito del comercio electrónico. Estos servicios deberán evitar, o al menos disminuir grandemente, el papel proactivo de los usuarios a la hora de realizar las búsquedas de la información o servicios solicitados.

Estos nuevos servicios de intermediación electrónica deberán ocupar el papel que tienen los actuales mercados en el comercio tradicional, es decir, un lugar/servicio que permita de una manera eficiente, escalable e integrada, donde los clientes encuentren a los suministradores, los suministradores encuentren a los clientes y se puedan realizar transacciones comerciales completas, con pago incluido y de una manera segura.

En este capítulo se expone la especificación realizada de un servicio de intermediación electrónica realizada en el ámbito del proyecto ABS (*Architecture for a Brokerage Information Service*, Arquitectura para un Servicio de Intermediación de Información), perteneciente al programa ACTS, el cual ha sido prototipado y validado en pruebas piloto internacionales durante este último año.

El objetivo de este capítulo es que, una vez conocido lo que se expone en ABS, y teniendo en cuenta los capítulos anteriores, se pueda llegar a concebir un sistema que sea capaz de gestionar la aplicación que se expone en las siguientes secciones, y que es precisamente la que da soporte a este servicio de intermediación.

El proyecto ABS está financiado parcialmente por la Comisión de la Unión Europea con el número AC206 del programa ACTS (*Advanced Communication Technologies and Services*, Tecnologías y Servicios de Comunicación Avanzados). El consorcio ABS está integrado por CNET (Francia), DeTeBerkom (Alemania), Telecom Finland (Finland), CET (Portugal), SEMA (Francia), VTT (Finlandia), NTUA (Grecia), DIT-UPM (España), Degriftour (Francia) y PoP (Alemania).

A continuación se da un repaso al concepto de servicio de intermediación electrónica, siguiendo con los actores que poseen un papel en dicho servicio. Más adelante se trata presentar algunas ideas de ABS, tales como los dominios planteados en el modelo de negocio y el modelo de representación del conocimiento, para finalmente describir la arquitectura de la aplicación de intermediación de ABS.

1.4.2. Servicio de Intermediación Electrónica

La base del comercio actual es el intercambio de bienes y servicios. Cuando un cliente necesita algo, crea una demanda en el mercado. Por otro lado, los proveedores proporcionan bienes y servicios, creando ofertas en el mercado. Para facilitar el encuentro de ambas partes, se requiere una "plataforma" que actúe como lugar de encuentro entre la oferta y la demanda. Esa plataforma la constituían tradicionalmente los mercados convencionales. Con la irrupción de las nuevas tecnologías de la información en los mercados surge el comercio electrónico. En dicho ámbito también hace falta algo similar que se plasma en las plataformas de mediación, presentadas a continuación.

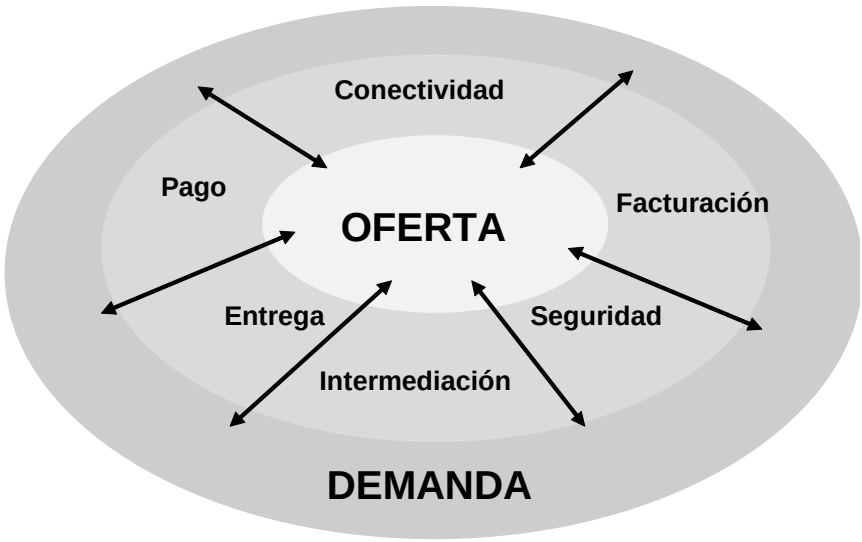


Figura 19. ***La oferta y la demanda***

El concepto de Intermediación Electrónica (*Electronic Brokerage*) se basa en la utilización de las nuevas tecnologías de la información para proporcionar un servicio que facilite y organice la relación entre la oferta y la demanda. Contará con dos clases de usuarios (usuarios finales y proveedores de bienes) y ofrecerá las siguientes funcionalidades a dichos usuarios:

- **Bajos costes** de búsqueda para ambos actores. La tarea principal de los intermediarios electrónicos será la búsqueda de una o varias ofertas que satisfagan una demanda o

viceversa. Es necesario limitar la complejidad de dichas búsquedas haciéndolas lo más trasparentes posibles para los usuarios.

- **Estructuración de las fuentes de información**: la eficiencia en las búsquedas de emparejamientos entre las ofertas y las demandas se basará en una estructuración adecuada de éstas en el intermediario. Esta estructuración permitirá a su vez una capacidad de navegación a través de las ofertas disponibles para los usuarios.
- **Combinación de la información**: el intermediario será capaz de combinar distintas ofertas o demandas para atender con una mayor calidad una petición de un usuario.
- **Presentación**: cuando las búsquedas nos proporcionen múltiples posibles soluciones, será necesario categorizar éstas para facilitar al usuario la toma de decisión.
- **Acceso uniforme**: independiente del tipo de la información, el intermediario proporcionará una visión uniforme de distintas ofertas y demandas heterogéneas.
- **Fiabilidad**: el intermediario velará por la calidad de las ofertas y demandas que maneja internamente, manteniéndolas actualizadas y preservando sus aspectos legales.
- **Confidencialidad**: cuando sea necesario el intermediario no desvelará la identidad de los distintos usuarios que intervienen en el servicio.

El consorcio ABS decidió utilizar para crear su plataforma de mediación los conceptos de modelado y desarrollo contenidos en la denominada *Arquitectura de computación de TINA-C* (*Telecommunications Information Networking Architecture Consortium*, Consorcio para la Arquitectura de Red de Information de Telecomunicaciones) [TINA95]. Dicha arquitectura ya fue definida en el capítulo dedicado a las Aplicaciones de Procesamiento Distribuido.

Lo más importante desde el punto de vista de la metodología de desarrollo empleada por el consorcio ABS, es que la arquitectura de computación define conceptos que han de ser utilizados para desarrollar aplicaciones dentro de la arquitectura de gestión, de red y de servicios. Todo *software* que se utilice dentro de TINA, ha de satisfacer los requisitos impuestos por los conceptos y principios contenidos dentro de la arquitectura de computación. No obstante, como se verá más adelante, esta no es la única influencia que el consorcio ABS ha recibido de la arquitectura de TINA-C.

La metodología de desarrollo adoptada por el consorcio ABS se basa en la utilización de las cinco perspectivas de ODP (particularizadas por la arquitectura de computación de TINA) y sus correspondientes lenguajes para especificar completamente la arquitectura del servicio de intermediación electrónica. Esas cinco perspectivas han sido igualmente

mostradas en el capítulo segundo, y son las perspectivas de negocio, información, computación, ingeniería y tecnología.

La especificación de un sistema de procesamiento distribuido bajo estas cinco perspectivas se puede equiparar a la aplicación de las fases de desarrollo típicas de la Ingeniería Software de la forma que indica la figura 20 [ITUT95].

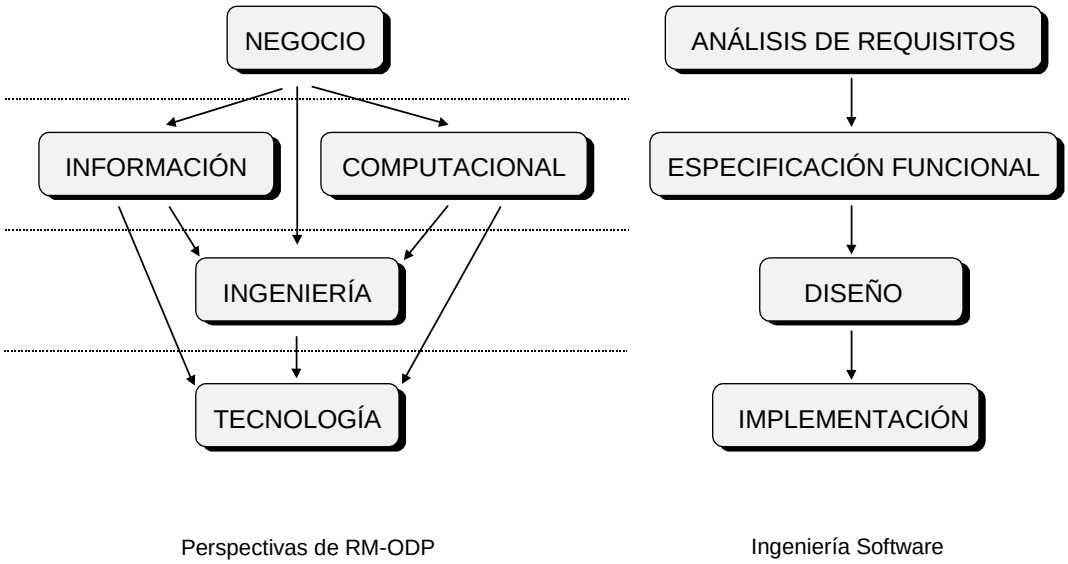


Figura 20. ***Relación entre las perspectivas de RM-ODP y las fases de desarrollo software***

1.4.3. El modelo de empresa del Servicio de Intermediación ABS.

El modelo de empresa del Servicio de Intermediación ABS [ABS96] [Tothenzan97], que se obtiene del análisis del servicio desde la perspectiva de empresa, describe la intermediación desde la perspectiva de qué actores están envueltos en sus operaciones y uso (sus papeles, sus requisitos, sus obligaciones...) y qué políticas se aplican en su comportamiento.

Dentro del modelo de empresa de ABS se ha identificado seis tipos de papeles de negocios y sus requisitos (Se muestran en la siguiente figura). Estos son:

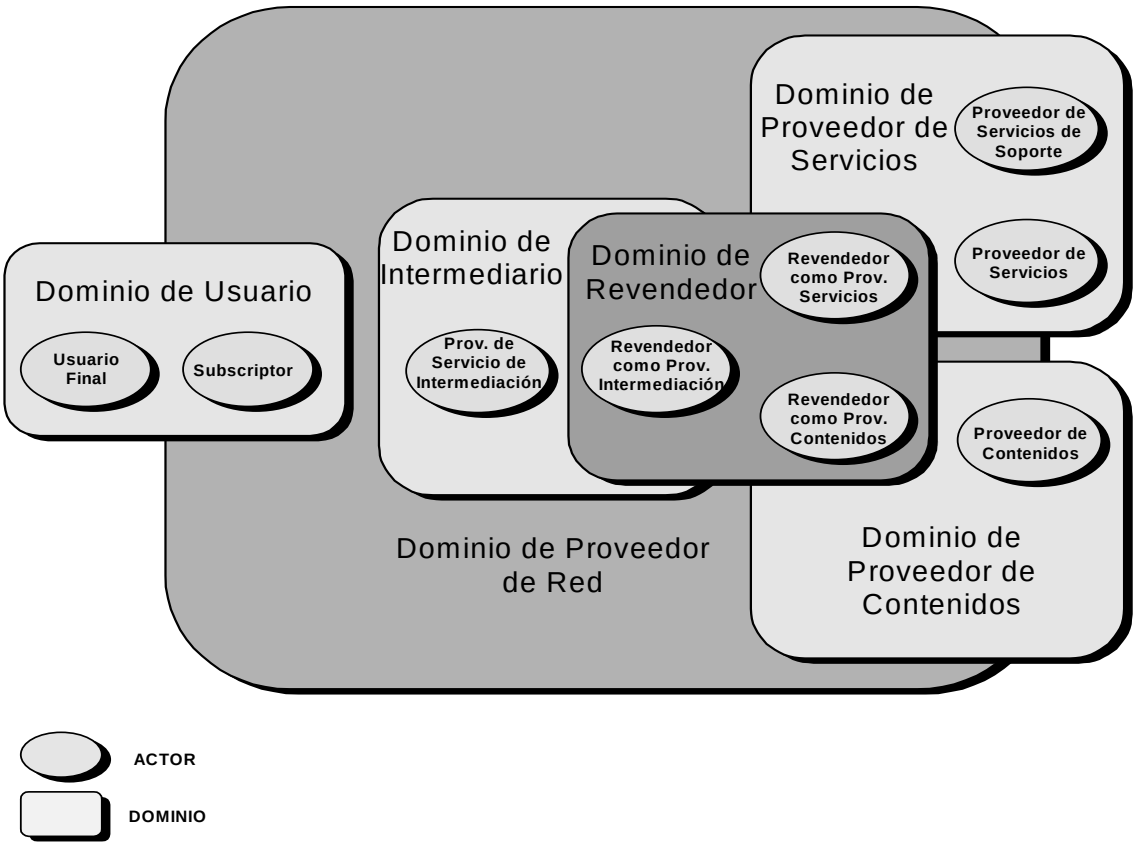


Figura 21. ***Actores.***

1. **Usuario del servicio de intermediación**: Es el papel de la entidad (ser humano, máquina o aplicación) que usa el servicio de intermediación para satisfacer sus necesidades. Este usuario puede ser un subscriptor o simplemente un usuario final, dependiendo de la relación contractual con el proveedor del servicio de intermediación, que se describirá a continuación.
2. **Proveedor del servicio de intermediación**: Es el papel de aquella entidad que ofrece información a los usuarios del servicio de intermediación de los servicios y contenidos que mantienen terceras partes. El proveedor del servicio de intermediación se puede considerar una extensión del tratante ODP [ODPTrading], pues traduce el concepto de confrontar servicios y bienes ofrecidos por proveedores con peticiones de la demanda.
3. **Proveedor del Servicio**: Es el papel de la entidad que proporciona servicios de telecomunicación, de información, o ambos, así como servicios de aplicación. Estos servicios se pueden dividir en dos grupos principales: Servicios que pueden ser el tema del servicio de intermediación (en el sentido de que pueden satisfacer algunas demandas) como vídeo bajo demanda, videoconferencia, etc., y servicios de soporte como, por ejemplo, seguridad, cobros, etc.

4. **Proveedor de contenidos**: Es el papel de aquella entidad que ofrece bienes o información negociables a la demanda o usuarios finales a través del servicio de intermediación.
5. **Revendedor**: Es una entidad cuyo papel es ofrecer servicios o contenidos a los usuarios del servicio de intermediación pero teniendo en cuenta que son servicios comprados a terceras partes. El revendedor realiza su actividad gracias a acuerdos contractuales con proveedores de servicios y contenido. Incorpora funciones de intermediación para ayudar a los usuarios a encontrar la oferta que pueda satisfacer sus requisitos de la mejor manera. En la figura 21 el solapamiento del dominio del revendedor con los otros es el resultado del hecho que el revendedor actúa como un proveedor de servicios y/o contenidos realizando las funciones de intermediario.
6. **Proveedor de Red**: Es el papel de la entidad que proporciona todas las funciones de interconexión a los otros actores. Se ha considerado un papel distinto, en vez de considerarlo un proveedor de servicio, pues no se trata de ofrecer su servicio a través del de intermediación y porque no es un servicio específico de mediación.

Con la identificación de los tipos de actores y sus requisitos y los correspondientes dominios administrativos, el modelo de negocio del servicio de intermediación se puede considerar suficientemente detallado como para proceder con el análisis de la arquitectura de dicho servicio.

1.4.4. Modelo de Información

El modelo de información trata de describir qué información está contenida en el intermediario electrónico y cómo ésta se procesa. Básicamente, el intermediario electrónico mantiene información acerca de los Usuarios del Servicio de Intermediación (el denominado Modelo de Información de Negocio), información que trata de modelar las diferentes áreas de negocio que se soportan (Dominio Conceptual), información sobre los contenidos de los proveedores (Dominio de Recursos) e información acerca de Proveedores de Servicio que proporcionan servicios auxiliares al servicio de intermediación (Modelos Auxiliares). Los Dominios de Recursos y los Dominios Conceptuales constituyen el denominado Modelo Núcleo del servicio de intermediación. Todos estos submodelos se pueden apreciar en la Fig. 4.

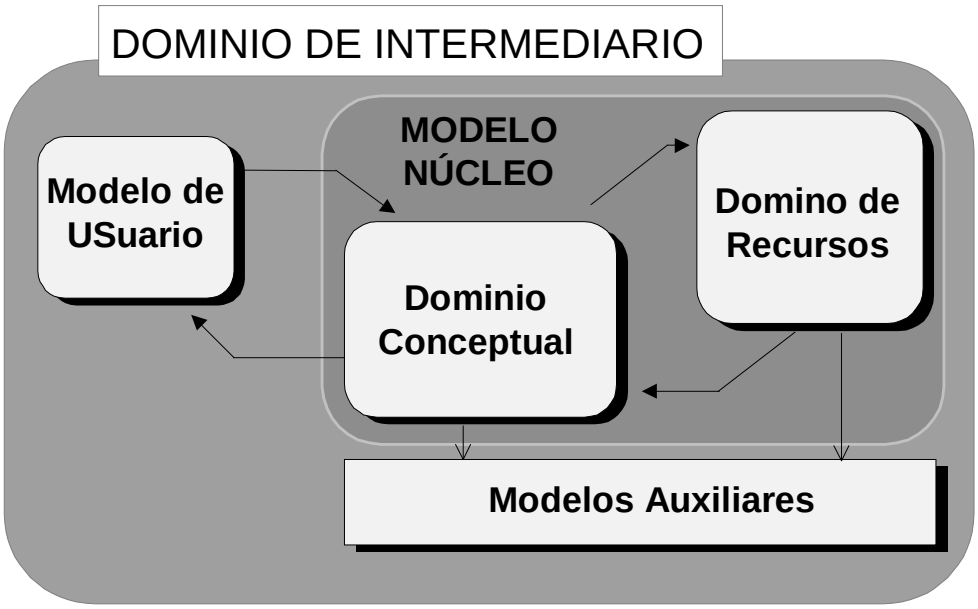


Figura 22. ***Relación entre los submodelos de información del servicio de intermediación electrónica***

Un aspecto muy importante es el modelo que utiliza el intermediario para representar el dominio de actuación de sus suministradores, es decir, el conocimiento que posee sobre los recursos que está intermediando. Para ello, se ha utilizado una representación del conocimiento denominada **Red conceptual**, basada en redes semánticas, que permite proyectar diversas perspectivas de un área u ontología en l que se ha denominado *viewpoint*.

La denominada Red Conceptual es parte del **Dominio Conceptual** y se puede definir como una estructura multi-relacional de nodos y relaciones entre nodos. Cada uno de esos nodos representa un concepto de la vida real. Un conjunto de estos conceptos, junto con las correspondientes relaciones, se puede proyectar en las denominadas Perspectivas que no son más que una abstracción de una particular área de negocio.

Esta representación es la base del funcionamiento del intermediario, ya que permitirá asociar los recursos de los suministradores a un concepto semántico que será utilizado para poder seleccionarlo ante las preguntas de los usuarios. Así mismo, el usuario tendrá la capacidad de navegar por esta representación del conocimiento de poder definir la información que necesita en el formato utilizado por el intermediario y así tener mayores posibilidades de éxito para seleccionar suministradores.

La figura 23 muestra un esquema de en qué consiste la Red Conceptual. En dicho esquema se puede apreciar la Red Conceptual (CN, *Conceptual Network*) en la parte superior. Los conceptos de la CN se agrupan de acuerdo a diferentes áreas de negocio. Cada grupo de

conceptos (cada área de trabajo) constituye una Perspectiva formada por Objetos Proyectados (SO, *Shadow Objects*). Un SO se podría definir como la proyección de un concepto sobre una Perspectiva. Un mismo concepto de la CN se podría proyectar sobre diferentes Perspectivas dando lugar a diferentes SO. De esa forma, por ejemplo, Santa Cruz de Tenerife se podría proyectar sobre la Perspectiva de *Vuelos en Europa* como un SO que hace referencia a posibles destinos. Del mismo modo, se podría proyectar sobre la *Perspectiva* de *Turismo Cultural en España* como un posible sitio a visitar. Los SOs de diferentes Perspectivas están relacionados por los denominados Puentes que permiten relacionar diferentes áreas de negocio (en el ejemplo, *Vuelos en Europa* con *Turismo Cultural en España*).

Asociados a los SOs se encuentran los Recursos (componentes del Domino de Recursos) que describen los contenidos y las características de los proveedores involucrados en el servicio de intermediación. Dicho de otro modo, los Recursos contienen información sobre cómo acceder a la información que puede satisfacer la demanda de los usuarios.

Este modelo de información cuenta con dos ventajas sobresalientes:

- Es un modelo general: las características de la Red Conceptual permiten añadir nuevas áreas de negocio de una manera muy sencilla.
- Es un modelo flexible: permite navegar por una determinada área de negocio, pasar de un área a otra mediante los Puentes, y, además de navegar, se permite realizar búsquedas a partir de las cuales el intermediario electrónico combinará diferentes fuentes de información para encontrar la respuesta que más satisfaga al usuario. El usuario podrá pasar del modo de navegación al de búsqueda o viceversa cuando así lo desee.

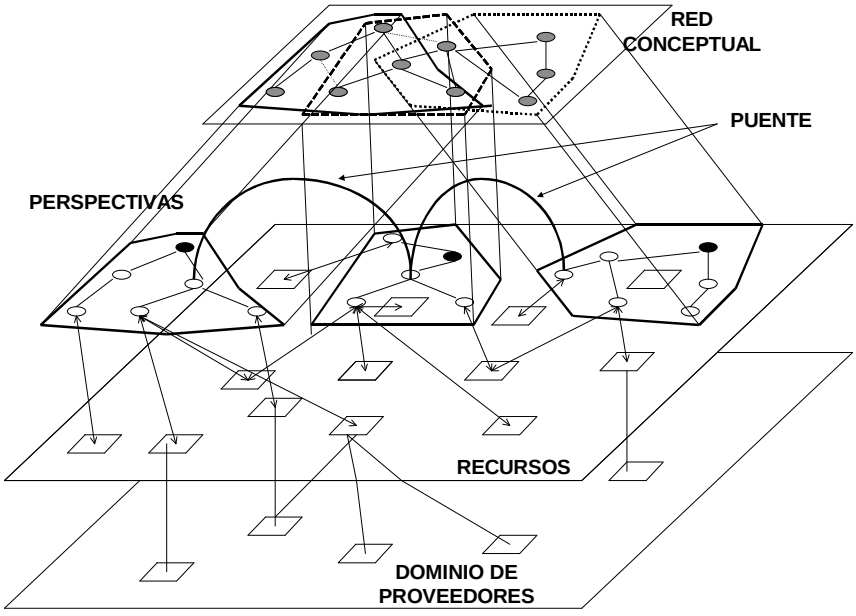


Figura 23. ***Representación del conocimiento***

1.4.5. Arquitectura del servicio

Los principales servicios proporcionados por el intermediario electrónico son los siguientes:

- Para un usuario
  - Capacidad de suscribirse y acceder al sistema
  - Capacidad de definir la información buscada, utilizando para su ayuda la representación del dominio que maneja el intermediario. Con esta definición el intermediario deberá encontrar los suministradores de información que se ajusten a dichas necesidades.
  - Capacidad de acceder a ofertas almacenadas directamente por los suministradores en el intermediario
- Para los suministradores.
  - Capacidad de suscribirse y acceder al sistema.
  - Capacidad de definir en el intermediario las capacidades del suministrador, es decir, el tipo de información que posee para que pueda ser seleccionado por el intermediario ante preguntas de los usuarios.
  - Capacidad de almacenar en el intermediario informaciones sobre ofertas específicas del suministrador.

Adicionalmente, existen las facilidades de gestión y operación del sistema y de federación entre distintos intermediarios, de manera que preguntas que alcancen un intermediario puedan ser redirigidas a otro intermediario en caso de que éste último sea identificado como potencial fuente de la información solicitada.

La arquitectura global del intermediario es la siguiente:

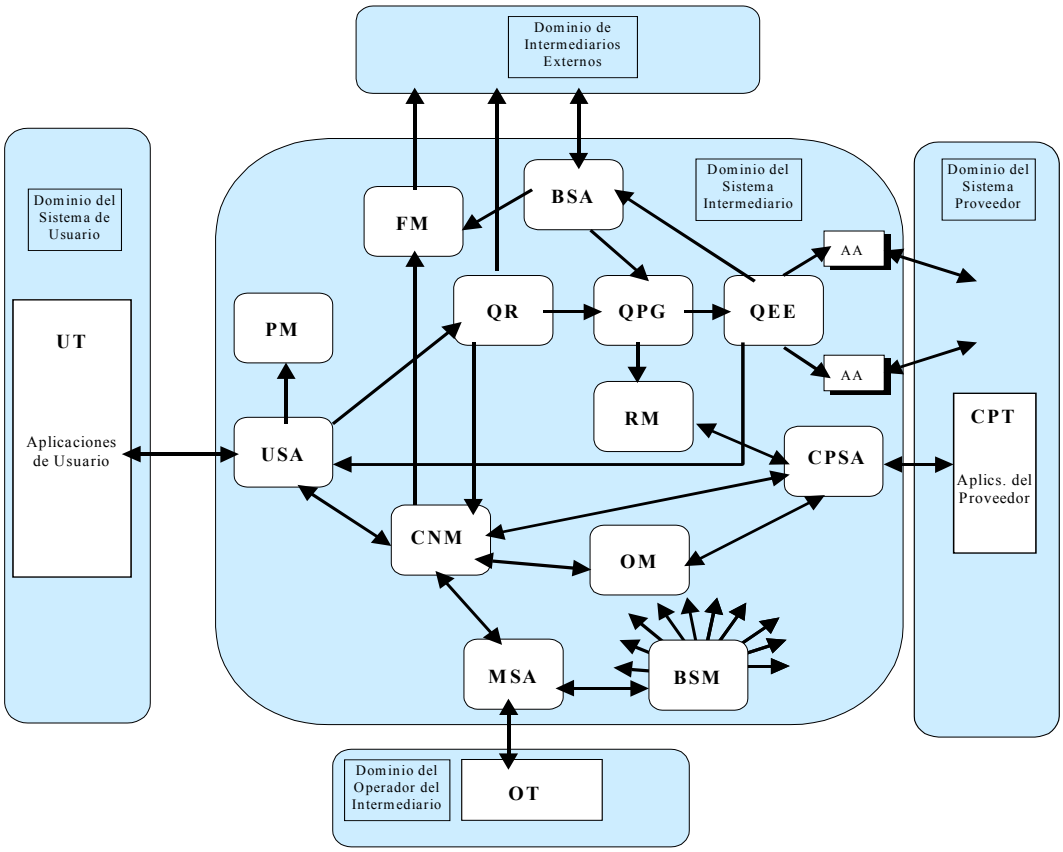


Figura 24. ***Arquitectura del sistema de intermediación***

Así mismo, la arquitectura de ABS de acceso a las sesiones se define de acuerdo a la arquitectura de TINA-C y se muestra a nivel general en la siguiente figura, donde los componentes del servicio se crean por los componentes de acceso después de una sesión de acceso con éxito.

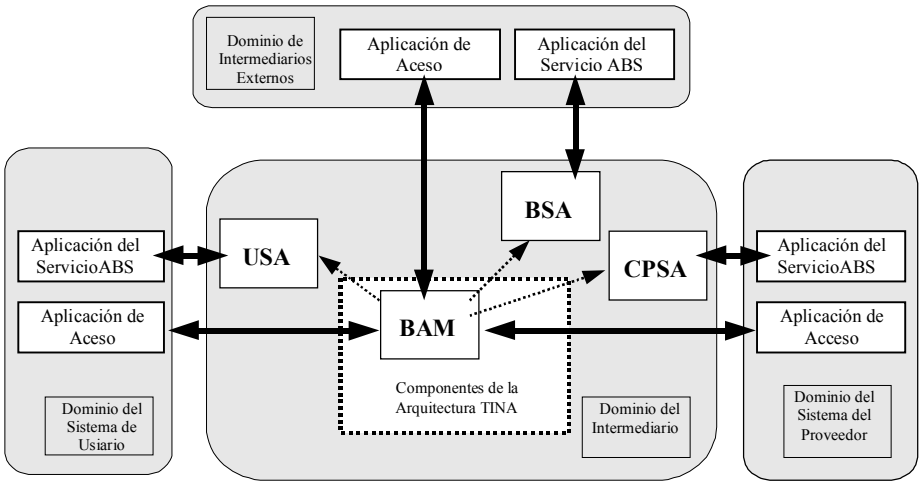


Figura 25. ***Arquitectura del sistema de acceso***

A continuación se expone el significado de los componentes de esta arquitectura, que ha sido prototipada sobre una plataforma CORBA y validada en el ámbito de unas pruebas pilotos internacionales. Estos componentes, integrantes de la aplicación de intermediación son justamente los que se quieren gestionar en el presente proyecto.

1. **Gestor de Accesos al Intermediario** (BAM, *Broker Access Manager*). Este bloque es responsable de completar el acceso a las sesiones del sistema ABS. Se define de acuerdo con la arquitectura TINA-C y es responsable de la autenticación del usuario y crear los agentes apropiados para el servicio a dar en la sesión. Los agentes de sesión del servicio ABS serán un Agente de Servicio de Usuario (USA, *User Service Agent*) para el acceso de usuarios privados, un Agente de Servicio de Proveedores de Contenidos (CPSA, *Content Provider Service Agent*) y un Agente de Servicio de Intermediación (BSA, *Broker Service Agent*) para acceso de intermediarios externos.
2. **Agente de Servicio de Usuario** (USA, *User Service Agent*). El Agente de Servicio de Usuario representa a los usuarios en el domino del intermediario. Es el punto de terminación del dominio del intermediario en una sesión de servicio con un usuario, y se crea tras una sesión de acceso exitosa con un usuario. Es accesible desde el dominio del usuario, sin tener en cuenta la localización. Es también responsable de decidir si la demanda del usuario es de navegación o de búsqueda, para consultar o actualizar el perfil del usuario, y para pedir la ejecución de la demanda a los bloques correspondientes.
3. **Agente de Servicio de Proveedores de Contenido** (CPSA: *Content Provider Service Agent*). El Agente de Servicio de Proveedores de Contenido representa a los proveedores en el dominio del Broker. Es el punto de terminación del dominio del intermediario de una sesión de servicio con un proveedor de contenidos (CP, *Content Provider*), y se crea tras una sesión de acceso con éxito con dicho CP. Se accede desde el dominio del CP y es también responsable de decidir si la petición del CP se refiere a la gestión de una oferta del CP, o una descripción de Recursos del Proveedor de Contenidos, o una actualización del perfil del proveedor de contenido.
4. **Agente de Servicio de Intermediación** (BSA, *Broker Service Agent*). El Agente de Servicio de Intermediación representa a un intermediario externo en el dominio del intermediario. Es el punto de terminación del dominio del intermediario en una sesión de servicio con un intermediario externo, y se crea tras una sesión de acceso exitosa. Se accede desde el dominio de un intermediario externo y es responsable de decidir si la petición del intermediario externo se refiere a exportar sus perspectivas o *viewpoints* o redirigir una consulta de usuario a su dominio.

5. **Redirector de Consultas** (QR, *Query Redirector*). El bloque Redirector de Consultas es responsable de redirigir una consulta de usuario al intermediario local, a otro(s) externo(s) o ambos.
6. **Generador de Plan de Consulta** (QPG, *Query Plan Generator*). El bloque Generador de Plan de Consultas se encarga de construir un plan de ejecución correspondiente a la consulta del usuario.
7. **Motor de Ejecución de Consultas y Agentes de Acceso** (QEE & AA, *Query Execution Engine and Access Agent*). El bloque Motor de Ejecución de Consultas se encarga de la ejecución del plan generado por el QPG. Los Agentes de Acceso son responsables de obtener la información de los proveedores de contenido.
8. **Gestor de Recursos** (RM, *Resource Manager*). El bloque Gestor de Recursos gestiona las descripciones de los Proveedores de Contenido (información sobre los contenidos de los proveedores).
9. **Gestor de Red Conceptual** (CNM, *Conceptual Network Manager*). El bloque Gestor de Red Conceptual se encarga de gestionar la red conceptual y las perspectivas.
10. **Gestor de Ofertas** (OM, *Offer Manager*). El bloque Gestor de Ofertas gestiona las ofertas del Proveedor de Contenidos (información sobre bienes específicos que el CP quiere registrar en el intermediario).
11. **Gestor de Federación** (FM, *Federation Manager*). El bloque Gestor de Federación es responsable de gestionar todas las actividades de importación o exportación de las perspectivas relacionadas con la funcionalidad de federación.
12. **Gestor de Perfiles** (PM, *Profile Manager*). El bloque Gestor de Perfiles se encarga de gestionar todos los perfiles de los actores que deben ser almacenados en el intermediario para completar de forma adecuada el servicio solicitado. Este bloque gestiona los perfiles correspondientes a usuarios finales, proveedores de contenido e intermediarios externos.
13. **Gestor del Servicio de Intermediación** (BSM, *Broker Service Manager*). El bloque Gestor del Servicio de Intermediación controla las interacciones entre el resto de los bloques funcionales y componentes relacionados. El BSM forma una parte importante del presente proyecto. El bloque BSM debería acceder a la información de gestión de los componentes del intermediario solicitados por el operador del intermediario, recibir notificaciones y alarmas de los componentes del intermediario, modificar componentes del intermediario de acuerdo a las peticiones del operador del intermedidario, asegurar el buen comportamiento de los componentes del intermediario cuando sea necesario y manejar procesos disparados en los componentes del intermediario. Así mismo, el bloque

Gestor del Servicio de Intermediación interactúa con todos los bloques funcionales para funciones de gestión y el MSA para obtener las peticiones del operador y devolver los resultados de dichas peticiones de gestión.

14. **Agente del Servicio de Gestión** (MSA, *Management Service Agent*). El Agente del Servicio de Gestión es el punto de contacto del intermediario con el operador del intermediario. Es accesible desde el dominio del Operador del Intermediación y es responsable de elegir el componente apropiado que ejecute la petición del operador.
15. **Terminal del Usuario Final** (UT, *Private User Terminal*). El bloque de la Terminal de Usuario se localiza en el Dominio del Usuario Final y es el componente utilizado por el usuario final para acceder al sistema de intermediación y solicitar los servicios de intermediación.
16. **Terminal del Proveedor de Contenidos** (CPT, *Content Provider Terminal*). El bloque Terminal del Proveedor de Contenidos se localiza en el dominio del proveedor de contenidos y es el componente que usa el proveedor de contenidos para acceder al sistema de intermediación y para solicitar los servicios del intermediario.
17. **Terminal de Operador** (OT, *Operator Terminal*). El bloque Terminal del Operador se localiza en el Dominio del Operador del Intermediario y es el componente que se usa por el operador para acceder al sistema de intermediación y para gestionar y administrar el mismo. La parte del OT encargada de la Gestión del Sistema (OT-MGR, *OT-Manager*) forma el otro gran bloque de este proyecto.

    El Terminal del Operador debería proporcionar las siguientes funcionalidades al Operador de Intermediación: Acceder al Sistema de Intermediación ABS, dar facilidades para visualizar el estado de todo el sistema, las alarmas o notificaciones del sistema e información específica de gestión solicitada por el operador, dar facilidades para hacer la configuración inicial de la Red Conceptual y también su evolución dinámica, dar facilidades para gestionar los actores cuando esto sea necesario, como, por ejemplo, la suscripción de intermediarios externos o proveedores de contenido, creación de perfiles, etc. El bloque Terminal de Operador interactúa con el MSA.

1.4.7. Conclusiones

En este capítulo se ha descrito un nuevo servicio de información avanzado que jugará un papel fundamental en el futuro Comercio Electrónico: el Servicio de Intermediación Electrónica que se está desarrollando en el contexto del proyecto ABS.

La utilización de las perspectivas definidas en el modelo de referencia de ODP, junto con los matices incorporados por la arquitectura de computación de TINA-C, se está mostrando como una metodología de desarrollo válida en el caso del servicio de intermediación electrónica de ABS.

Tras la implementación de un primer prototipo basado en tecnologías como Java y CORBA, se ha llevado a cabo una serie de pruebas de campo en Portugal, Francia, Finlandia y Alemania con usuarios y proveedores reales en diferentes áreas de negocio. Los resultados de dichas pruebas han servido para refinar la primera versión de la arquitectura de tal forma que se han incorporado nuevos servicios de valor añadido que se han visto deseables en el servicio de intermediación electrónica.

La gestión de la aplicación de intermediación resulta de gran ayuda a la hora de ejecutar el sistema, tanto en las pruebas antes mencionadas como en un caso real, pues permiten al operador encargado del sistema ver como está respondiendo el sistema en cada momento. En la segunda parte de esta memoria se desarrollará el sistema que se encarga de la gestión de esta aplicación distribuida de intermediación.

2. Desarrollo del Proyecto

2.1. Metodología

2.1.1. Introducción

Tras haber estudiado en los capítulos anteriores temas que trataban de dar una visión de la gestión de red, aplicaciones distribuidas, y posibles formas de integración, este capítulo pretende describir la metodología empleada para desarrollar la gestión de una aplicación distribuida. Una vez se ha estudiado los fundamentos teóricos precisos para tener los conocimientos necesarios a la hora de abordar la fase de desarrollo, se comienza dicho desarrollo. Para llevar a cabo el mismo han sido necesarias un conjunto de técnicas que se pretenden esbozar en este capítulo y que se ampliarán en los siguientes.

El capítulo se divide en las siguientes partes:

1. **Metodología de desarrollo**. Trata de describir la metodología empleada para realizar el desarrollo.
2. **Lenguaje de Modelado Unificado (UML, *Unified Modelling Language*)**. Da una visión rápida a este lenguaje de modelado de objetos, usado para describir el modelo del sistema a desarrollar.

Tras estas secciones se puede encontrar un apartado de conclusiones que trata de resumir las técnicas expuestas a lo largo del capítulo.

2.1.2. Metodología de desarrollo

Para modelar los sistemas *software* se han empleado hasta ahora diversas orientaciones, cada una clasificada de acuerdo al enfoque de la abstracción (qué es lo básico del problema: procesos, comportamiento, datos), al soporte de la descripción (los tipos de modelo a utilizar), a los mecanismos de partición del problema (relaciones entre modelos) y a la transición entre etapas de desarrollo.

Algunas tecnologías utilizadas son el desarrollo estructurado o funcional, el diseño de sistemas de control, el diseño de bases de datos y la orientación a objetos. Las tecnologías orientadas a objetos tienen como características principales que su enfoque se centra en el concepto de objeto, como una visión conjunta de procesos, comportamientos y datos, que tiene como soporte modelos de objetos, junto con modelos de otras tecnologías (diagramas de flujos de datos, diagramas entidad-relación, de estados y transiciones, etc.), que se parte en

perspectivas estructural, dinámica y funcional, y que permite una transición gradual, aumentando el nivel de detalle.

Existen distintas metodologías para llevar a la práctica una determinada tecnología *software*. Ofrecen notaciones y modelos de descripción concretos que permiten representar el sistema según unas determinadas reglas, así como guías metodológicas sobre cómo realizar el proceso de desarrollo y herramientas *software* de ayuda al proceso. Últimamente estos modelos se han unificado en lo que es el UML (*Unified Modelling Language*, Lenguaje de Modelado Unificado). UML es el lenguaje que se ha empleado para el modelado del sistema de gestión, mediante la herramienta Rational Rose98 [Rational98]. Sin embargo, UML no propone una metodología de desarrollo específica.

A la hora de abordar el desarrollo del sistema de gestión se tomaron ideas de una metodología simple, orientada a objetos, que se sugiere en [Harmon98]. Dicha metodología se conoce como iterativa, cíclica o en espiral. Implica esencialmente que se desarrollan las aplicaciones mediante aproximaciones sucesivas. Primero se desarrolla el núcleo de la aplicación, que será el prototipo inicial. Después, se refina el prototipo mejorándolo y extendiéndolo. Dependiendo de la complejidad de la aplicación, esto puede tomar varias iteraciones. Tras cada ciclo aparece un prototipo hasta que finalmente se llega a la aplicación deseada.

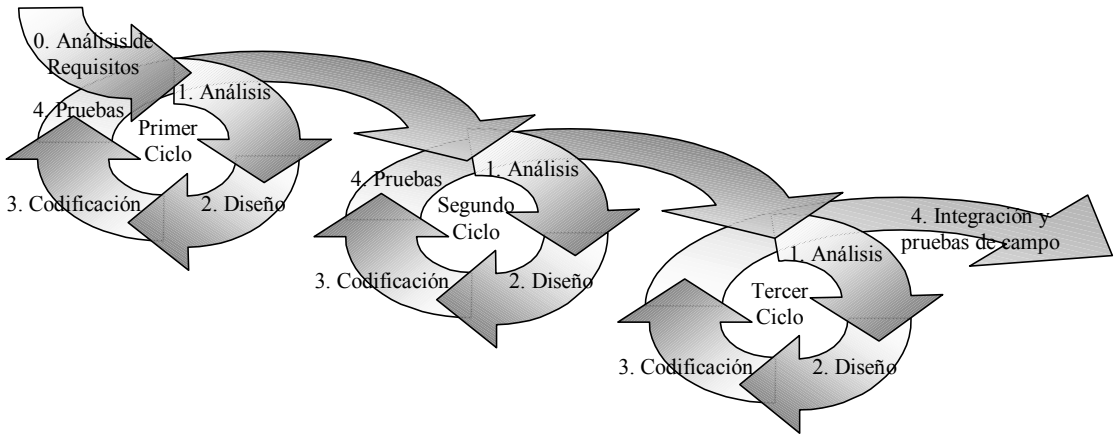


Figura 26. ***Ciclo de vida en espiral***

Así mismo, durante las fases de implementación se han tomado ideas del método de desarrollo con ciclo de vida en V [Pressman96], que muestra como el paso entre fases de desarrollo siempre implica una fase de pruebas.

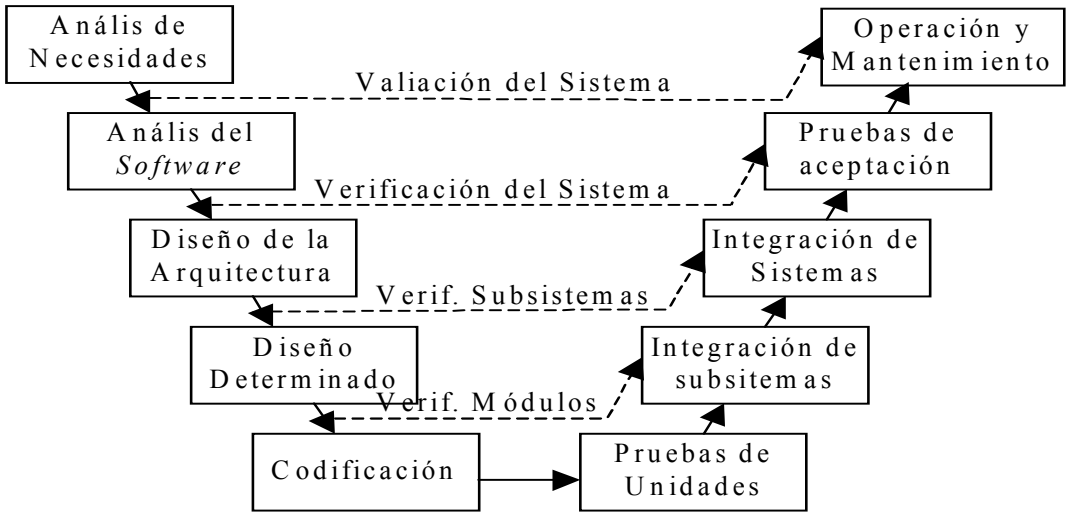


Figura 27. ***Ciclo de vida en V***

2.1.2.1. Fases de un ciclo de desarrollo.

Cada ciclo comienza definiendo el ámbito y requisitos de la totalidad del proyecto, dando una descripción de alto nivel de la aplicación. Se debe documentar qué hará el sistema, quién lo usará y qué restricciones o criterio de rendimiento necesitará para que se considere satisfactorio. Posteriormente se esboza un modelo de objetos de alto nivel con todos los objetos que se requerirán para la aplicación.

Siguiendo esto, una vez se tiene una idea clara del ámbito concreto de la aplicación, se selecciona una parte de la misma para desarrollarse primero. Los criterios para elegir esta parte son variados, pero para sistemas grandes se suele desarrollar primero un prototipo del sistema que no realice grandes funciones, pero que demuestre que puede actuar con el resto de la infraestructura con quién tendrá que tratar. Esto es para asegurar que el diseño en grandes rasgos es el adecuado.

Una vez desarrollada una primera aproximación, se debe identificar los problemas que necesitan una revisión del diseño. Así mismo, a la hora de empezar, se debe planear los ciclos que deben llevar a la aplicación final.

Una vez se ha hecho un estudio de los requisitos iniciales, se realiza una fase inicial de análisis, en la cual se subdivide el sistema y selecciona una parte del mismo para desarrollar en el primer ciclo.

Durante el análisis el trabajo se enfoca en diseñar un sistema lógico o ideal que satisfaga los puntos requeridos sin pensar en consideraciones de implementación. Durante el diseño se modifica el trabajo teniendo en cuenta restricciones del mundo real. Durante la fase

de codificación se escribe el código. Entonces se comprueba el prototipo para ver que satisface las necesidades del usuario o cómo funciona en el entorno donde se piensa instalar.

Tras el primer prototipo se continúa con la segunda fase de análisis, en la que se comienza reexaminando el análisis primero basándose en lo aprendido en el desarrollo del primer prototipo. Normalmente se hacen algunas modificaciones, aunque a veces se decide rediseñar por completo. Después, se expande la funcionalidad del primer prototipo, siguiendo de nuevo el ciclo de diseño, codificación y pruebas. Y así, tantas interacciones como sean necesarias.

En los siguientes apartados se completará la definición de las distintas fases, incluyendo, cuando se crea necesario los diagramas UML que son convenientes utilizar. Se recuerda que la segunda parte de este capítulo se dedica por completo a este lenguaje, por lo que no debe haber ningún problema en la lectura.

2.1.2.2. Fase de Requisitos

El primer ciclo de desarrollo es único, pues comienza con una fase de requisitos que precede a la fase de análisis. Es, por tanto, una especie de puerta de entrada al desarrollo. Se puede llevar a cabo de distintas maneras, pero en cualquier caso el estado inicial acaba con la especificación de requisitos de alguna clase, es decir, un documento que proporcione una descripción general de la aplicación a desarrollar.

Métodos posibles son estudios de Reingeniería de Procesos o bien la creación de escenarios y casos de uso.

Una buena especificación de requisitos debe proporcionar una descripción lo suficientemente completa de la aplicación a desarrollar para permitir a los desarrolladores considerar como se puede modularizar la aplicación. El IEEE ha propuesto un estándar indicando lo que debe contener una especificación de requisitos [IEEE84].

2.1.2.3. Fase de Análisis

En las metodologías tradicionales en cascada, se solía insistir en que se mantenía una distinción rígida entre análisis y diseño. La fase de análisis consistía en pensar que debía hacer la aplicación. La de diseño, en decidir como hacerlo. Manteniendo análisis y diseño separados, se mantiene las distintas opciones abiertas y se evita situaciones en las que se crea sistemas llenos de impedimentos debido a las restricciones impuestas por el diseño del sistema.

Sin embargo, en el mundo real la frontera entre análisis y diseño no está tan clara. Es difícil mantener el análisis en lo abstracto. Además, las situaciones reales siempre imponen restricciones que se deben tener en cuenta antes de empezar la codificación. En el mundo de la orientación a objetos, esta frontera resulta aún más difusa [Harmon98].

Una de las ventajas de la aproximación en espiral es que pasa del análisis al diseño, y al comenzar el siguiente ciclo se vuelve al análisis. Esto permite a los desarrolladores reconsiderar lo que se ha hecho durante la fase de diseño y depurarlo.

En esta fase se puede utilizar cinco grupos de diagramas UML. Cada uno de los diagramas básicos se puede extender para particularizar algunos problemas. Alguno de ellos, el de clases y el de interacciones, se reutilizan durante la fase de diseño. Otros sólo se usan en casos particulares. En algunos casos se usan diagramas de paquetes tras analizar los casos de uso. En otros, no se consideran hasta la fase de diseño.

2.1.2.4. Fase de Diseño

La fase de diseño comienza cuando se cambia el enfoque de las relaciones lógicas entre objetos al esbozo físico del sistema. A diferencia de otras metodologías, en las que normalmente hay que cambiar totalmente de notación, la notación orientada a objetos no cambia cuando se pasa del análisis al diseño.

En el diseño simplemente se extiende el diagrama usado en el análisis y se añade información. Se aumenta el nivel de especificidad sobre las operaciones que llevan a cabo determinadas tareas. Además, se toman decisiones concretas sobre el borrador del sistema: Qué *hardware* se usará, cómo se modularizará el *software* y qué módulos se colocarán en qué máquinas.

En todos los desarrollos orientados a objetos está la cuestión de la reutilización de clases. Esto, sin embargo, no es nada trivial, dado que hay que pensar en el desarrollo de clases reutilizables mientras se está tratando todavía de buscar qué clases desarrollar. En cualquier caso se suele usar clases y componentes ya definidos, como el *Java Developement Kit* (JDK, Herramienta de Desarrollo Java) [Macaray96].

2.1.2.5. Fase de Codificación

La siguiente fase en el ciclo de desarrollo comienza cuando se empieza a escribir el código. Durante el diseño se habrá especificado el conjunto de objetos del sistema, y se habrá identificado probablemente todos los atributos y la mayor parte de las operaciones. Incluso se puede haber escrito pseudo-código para especificar que hará cada operación.

En el desarrollo orientado a objetos, las estructuras y nombres de clases creados en las fases de análisis y diseño se convierten en elementos de código en la de codificación. Una vez más la transición entre dos fases se hace de forma suave. Únicamente se dice como se lleva a cabo una operación expresando las operaciones en un lenguaje orientado a objetos.

Durante estas fases puede generar código, incluyendo todos los nombres y estructuras de clases, tipos de datos y otra información que se haya especificado sobre los atributos y operaciones. Se necesita completar el código requerido realmente para implementar cada operación y escribir cualquier código específico para interfaces o enlaces.

2.1.2.6. Fase de Pruebas

La fase final del ciclo envuelve probar el código que se ha desarrollado para ver si funciona adecuadamente. Dependiendo de la naturaleza del primer prototipo, las pruebas será un trámite formal o bien significará pruebas de campo en las que el sistema interactuará con los usuarios.

En un ciclo de desarrollo iterativo, tras las pruebas se vuelve al análisis y diseño. Tras probar el prototipo y de acuerdo a los problemas que se haya encontrado, se cambia el modelo original o se comienza a expandir y mejorar, comenzando un nuevo ciclo.

Las pruebas que se realicen serán de varios tipos según el nivel al que afecten del ciclo de vida, tal como se aprecia en la figura 27. Estas pueden ser unitarias, de integración y de sistema.

Las pruebas unitarias se realizan sobre los módulos codificados y pueden ser de dos tipos:

El primero de ellos son las pruebas de **caja blanca o transparente**. Este tipo de pruebas se basa en el conocimiento de la estructura interna del programa. Se basa en ir viendo como se va ejecutando el código, probando que el programa recorre todos los caminos posibles de ejecución. Para ello se subdivide la tarea probando:

1. La estructura de datos y las funciones realizadas sobre ellas.
2. Las decisiones lógicas.
3. Los bucles, viendo que funcionaban para un número pequeño y grande de vueltas.
4. El conjunto de caminos básicos.

El segundo tipo de pruebas es de **caja negra u opaca**. Este tipo de pruebas se realiza sin tener en cuenta la estructura interior. Es recomendable que este trabajo lo hiciera alguien distinto al desarrollador. Las pruebas se basaron en la especificación. Para una mayor cobertura se descompone el espacio de datos en clases de equivalencia. Por ejemplo, para una

interfaz de gestión se puede probar que se puede realizar las funciones de tipo *get*, *set*, y *getnext* sobre todas aquellas variables que sean accesibles.

Las pruebas de integración se deben a que un sistema *software* no es la suma de sus módulos. Puede haber incompatibilidades de interfaz o de interacción y también problemas por un alto acoplamiento de los datos.

Se puede integrar desde diversos enfoques: Ascendente, descendente e intermedio o *sandwich*, según se implemente los módulos clientes o los servidores, haciendo uso de clientes y servidores falsos durante el proceso, hasta llegar a la integración total.

Para finalizar el proceso de pruebas, se realizan aquellas que se refieren al sistema global. Se ejecuta el sistema procediendo a utilizar las herramientas y aplicaciones implementadas. Entre estas pruebas están:

1. Pruebas de recuperación: Ante una caída, el sistema se recupera.
2. Pruebas de seguridad: Probar accesos no permitidos
3. Pruebas de resistencia: El sistema no se degrada ante un funcionamiento prolongado.
4. Pruebas de rendimiento: Se mide el rendimiento del sistema.

2.1.3. Lenguaje de Modelado Unificado

Este apartado pretende explicar qué es UML (*Unified Modelling Language*, Lenguaje de Modelado Unificado) y para qué se ha empleado en el análisis y diseño del sistema de gestión de ABS. UML es un lenguaje para especificar, visualizar, construir y documentar el desarrollo de sistemas *software*, así como para el modelado de negocios y otros sistemas que no son de *software*. UML representa una colección de las mejores prácticas de ingeniería que han probado tener éxito en el modelado de sistemas grandes y complejos. Sin embargo, como ya se adelantaba, UML no da una metodología de desarrollo. Sólo define unos diagramas (es un lenguaje de modelado como su propio nombre indica), cada uno de ellos más o menos útil para determinadas tareas. El estándar no dice cómo utilizarlos durante todo el ciclo de desarrollo, pero hay muchas propuestas de metodologías utilizables según el contexto en que se utilice.

UML lleva recibiendo galardones desde hace algún tiempo y se ha aprobado como estándar del Object Management Group (OMG, *Grupo de Gestión de Objetos*), creador de CORBA. Además, se ha usado en la práctica, y tiene algunos éxitos memorables. Hay algunas áreas en las que todavía se requiere un esfuerzo mayor, incluyendo la falta de un proceso UML, herramientas compatibles con UML y la posibilidad de obtener un diseño completo hasta la codificación e implementación.

Una de las metas principales del diseño en UML es proporcionar a los usuarios un lenguaje de modelado preparado para ser usado y visualmente expresivo con el que se pueda desarrollar e intercambiar modelos significativos. Por esta razón, la metodología UML se ha elegido para describir los requisitos del sistema de gestión de ABS.

Así, para obtener la especificación de la arquitectura de ABS, la aproximación a seguir ha sido definir primero diagramas de casos de uso UML, modelar los requisitos globales y entonces definir los diagramas de interacción, que modelen el comportamiento dinámico del sistema de gestión ABS en términos de la interacción entre los componentes de ABS (o clases, de acuerdo con la terminología UML).

2.1.3.1. Símbolos UML

Antes de presentar los diagramas UML, puede resultar de ayuda considerar la naturaleza de los símbolos que se usan en ellos. Los símbolos UML son de tres tipos básicos: símbolos nucleares (*core*) o de primera clase; extensiones especializadas de los primeros, que se llaman adornos (*adornments*); y símbolos muy especializados que se denominan estereotipos. A su vez, los símbolos nucleares se dividen en dos grupos: símbolos de modelado y de relación.

La siguiente figura ilustra el párrafo anterior. Es conveniente hacer notar que clase, estado y caso de uso son elementos nucleares del modelo y asociación y dependencia son relaciones nucleares. Se puede adornar una clase añadiendo información sobre sus atributos y operaciones. Si se quiere usar versiones de clases realmente especializadas, se está en el rango de estereotipos como por ejemplo clases de control y entidad. De una forma similar, una línea recta es una asociación. Colocando un rombo en la línea —un adorno— indica que la asociación es una agregación, y poniéndole un número se indica cuantos ejemplares están envueltos en la relación.

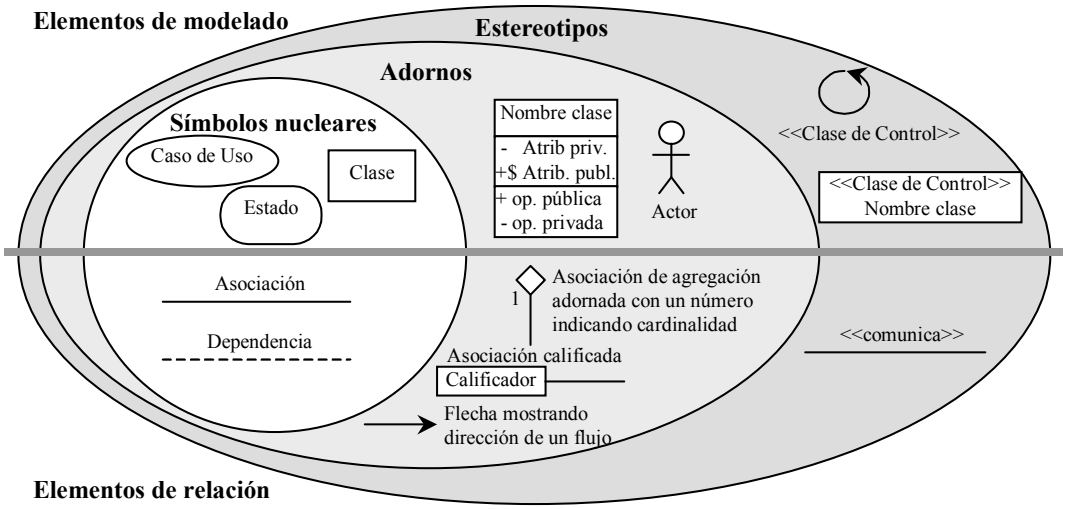


Figura 28. ***Elementos de UML***

2.1.3.2. Principales componentes y diagramas

En este apartado se quiere incluir aquellos componentes y diagramas en los que se basa principalmente UML para atacar las fases de análisis y diseño del modelado de un sistema.

Es conveniente empezar con las **Clases**. Una clase captura la estructura y el comportamiento común de un conjunto de objetos. Es una abstracción de elementos del mundo real. Cuando estos elementos existen en el mundo real, son ejemplares de una clase y son referidos como objetos. Para cada clase se puede crear un diagrama de estados que describa dicho comportamiento.

Una clase se representa como un cuadro con tres partes, con su nombre en la parte superior, una lista de atributos en la intermedia y una lista de operaciones en la inferior. Los atributos y operaciones del cuadro-clase se pueden obviar para reducir detalles para ver un esquema general. El ocultar una de estas secciones no implica la ausencia o no de atributos u operaciones, pero dibujar una parte vacía explicita que no hay elementos en dicha parte.

Una **Interfaz** especifica las operaciones visibles externamente de una clase y/o componente, y no está implementado en sí mismo. Una interfaz especifica típicamente una parte limitada del comportamiento de una clase o componente.

Una interfaz se representa como un pequeño círculo con una línea al componente que conceptualiza la interfaz, o bien, un cuadro de tres partes, al estilo de una clase, pero con el estereotipo de interfaz.

Un **Diagrama de Clases** es un gráfico que describe de forma genérica posibles sistemas. Los diagramas de clase contienen iconos representando clases, interfaces y sus relaciones. Se pueden usar en el proceso de análisis, mostrando los papeles y responsabilidades de las entidades que proporcionan el comportamiento del sistema, y en el de diseño, capturando la estructura de las clases que forman la arquitectura del sistema.

Los diagramas de clases se han usado ampliamente durante la fase de diseño del sistema de gestión desarrollado en este proyecto.

Un **Paquete** es un mecanismo de agrupación que se puede usar para designar tanto agrupaciones lógicas como físicas, pero que también se puede usar para designar agrupaciones de casos de uso, entre otras. Una agrupación de casos de uso es un paquete de casos de uso.

Los **Paquetes Lógicos** sirven para partir el modelo lógico de un sistema. Son grupos de clases altamente relacionadas que son mutuamente cohesivas, y con un grado bajo de acoplamiento con otros grupos. Se usan para agrupar clases, interfaces y otros paquetes.

Se pueden representar únicamente en diagramas de clases. Esto permite expresar y preservar elementos arquitecturales importantes del diseño del sistema. Se puede capturar un diseño de alto nivel del sistema creando un diagrama de clases que consista únicamente en paquetes.

Se representan como un cuadro con una solapa, y contienen en su interior el nombre del paquete.

Se puede establecer relaciones de dependencia entre paquetes. Este tipo de relación implica que las clases contenidas en el paquete cliente pueden heredar, contener ejemplares y hacer uso, entre otras, de las clases que son exportadas por el paquete suministrador.

Durante la fase de análisis se esbozó un diagrama de paquetes, que después se han definido en mayor profundidad en la fase de diseño.

Los siguientes apartados tratan diagramas que han sido usados extensamente en este proyecto. Es por ello por lo que se les da un tratamiento especial.

2.1.3.3. Diagramas de casos de uso

A la hora de definir los requisitos del sistema de gestión, la especificación ha hecho uso de los diagramas de casos de uso, que poseen los siguientes componentes UML: Actores y casos de uso, relacionados entre sí.

Un **actor** es un adorno de una clase y se puede representar como un icono especial en diagrama de casos de uso. Modela una clase de objetos fuera del dominio del sistema, la cual

interactúa directamente con el sistema, y define un conjunto coherente de papeles que pueden tomar los usuarios de una entidad cuando interactúan con dicha entidad. Cada actor tiene un papel para cada caso de uso en el que se involucra

Los actores se representan normalmente como una figura humana. Cada actor debe tener un nombre. Cada aspecto de un icono de actor en un diagrama representa un subconjunto del modelo de información sobre dicho actor. Un actor también puede poseer un conjunto de interfaces, cada una de las cuales describe como otros elementos se pueden comunicar con el actor.

Así mismo, un actor puede heredar de otros actores. Esto supone que el actor heredero podrá jugar los mismos papeles que el actor heredado, es decir, comunicarse con el mismo conjunto de casos de uso, como si del actor heredado se tratara.

Puesto que un actor es externo a la entidad, su estructura interna no está definida, sino únicamente su exterior, tal y como lo ve la entidad. Los ejemplares de actores se comunican con las entidades enviando y recibiendo ejemplares de mensajes a y de los ejemplares de casos de uso, y a un nivel de consciencia, a y de objetos. Esto se expresa por asociaciones entre el actor y el caso de uso o clase.

Un **caso de uso** es una secuencia de transacciones realizadas por un sistema en respuesta al disparo de un evento iniciado por un actor para el sistema. Contiene todos los eventos que pueden ocurrir entre un par actor-caso de uso, y no solamente aquellos que ocurren en un escenario particular. Contiene además un conjunto de escenarios que explican varias secuencias de interacciones dentro de la transacción.

La construcción de casos de uso se emplea para definir el comportamiento de un sistema u otra entidad semántica sin revelar la estructura interna de dicha entidad. Cada uno especifica una secuencia de acciones, incluyendo variantes, que puede realizar la entidad, interactuando con actores de la entidad. Las acciones incluyen cambios de estado y comunicaciones con el entorno del caso de uso. Un caso de uso también incluye posibles variantes de esta secuencia. Por ejemplo, secuencias alternativas, comportamientos excepcionales, manejo de errores, etc. El conjunto completo de casos de uso especifica todas las formas distintas para utilizar la entidad, es decir, todos los comportamientos de la entidad se expresan mediante sus casos de uso. Estos se agrupan en paquetes según convenga.

El aspecto normal de un caso de uso es una elipse, que debe tener un nombre. Suele estar escrito como un texto de descripción informal de los actores externos y la secuencia de eventos entre los objetos que forman la transacción. Los nombres de los casos de uso suelen empezar con un verbo. El nombre se presenta bajo el icono. Cada aspecto de un icono de caso

de uso en un diagrama muestra un subconjunto de la información del modelo sobre el caso de uso.

Puede haber asociaciones entre casos de uso y los actores de dichos casos de uso. Serán aquellas asociaciones de estados que ejemplaricen a un caso de uso y a un usuario jugando uno de los papeles del actor comunicándose con otro.

Además, los casos de uso se pueden relacionar con otros simplemente mediante relaciones de **extensión** y **uso**. Una relación de extensión denota la extensión de la secuencia de un caso de uso con la secuencia de otro, mientras que la relación de uso denota que los casos de uso comparten un comportamiento común.

La consciencia de un caso de uso se puede especificar por un conjunto de **colaboraciones**, es decir, ejemplares del sistema que interactúan para realizar la secuencia del caso de uso.

Considerando las definiciones previas, un **diagrama** de casos de uso presenta una colección de casos de uso y actores y se usa típicamente para especificar o caracterizar la funcionalidad y comportamiento del conjunto de un sistema o aplicación interactuando con uno o más actores externos.

Los Actores son los usuarios y cualquier sistema externo que pueda interactuar con el sistema. Ya que los actores representan a usuarios del sistema, ayudan a delimitar el sistema y dar una visión más clara de que se supone que debe hacer. Los casos de uso se desarrollan en la base de que son necesarios para los actores. Esto asegura que el sistema será lo que el usuario espera de él.

Los diagrama de casos de uso contienen iconos que representan actores, relaciones de asociación, de generalización, paquetes y casos de uso. Se puede crear un caso de uso de primer nivel para ver el contexto de un sistema y los límites de comportamiento del sistema. También, para describir una parte de un sistema. Los casos de uso pueden incluir otros casos de uso que sean parte de su comportamiento. Un diagrama de casos de uso muestra el conjunto de actores externos y los casos de uso del sistema en los que los actores participan.

2.1.3.4. Diagramas de Interacción

Un diagrama de interacción es la concepción de un caso de uso. Es típicamente un recorrido por el que fluyen los eventos del caso de uso. Los diagramas de interacción contiene objetos y mensajes entre los objetos que muestran cómo se conceptualiza el comportamiento.

Muestra una interacción ordenada en secuencias de tiempo. En particular, muestra los objetos participando en la interacción, siguiendo sus caminos de ejecución y los mensajes que

intercambian ordenados en una secuencia de tiempo. No muestra las asociaciones entre los objetos.

Puede existir de una forma genérica (describe todas las posibles secuencias) y en una forma ejemplarizada (describe una secuencia real consistente con la forma genérica). En casos sin bucles o saltos, las dos formas son isomórficas. Tiene dos dimensiones: La vertical representa el tiempo, y la horizontal, los objetos.

Se usan para encontrar los objetos necesitados por un sistema y, dentro del contexto del proyecto ABS, para hacer las especificaciones de la arquitectura de los diversos susbsistemas que lo componen.

La aproximación para la definición de los diagramas de interacción se basa en la estructura de los casos de uso. Para cada uno de ellos, se presentará un diagrama de interacciones representando un escenario específico del caso de uso.

2.1.3.5. Diagramas y Fases

A continuación, y tratando de relacionarlo todo, se presenta una tabla que pretende relacionar los diagramas con las fases en que se han utilizado.

Tabla 1. ***Diagramas y fases.***

Diagramas empleados	Fases del desarrollo
Clases	Diseño
Paquetes	Análisis y Diseño
Casos de Uso	Análisis
Interacción	Análisis y Diseño

2.1.4. Conclusiones

Este capítulo ha tratado de mostrar la metodología empleada para el desarrollo del sistema de gestión de ABS. Así, primero se ha presentado una metodología simple de ciclo de vida en cascada, en la que se ha empleado como lenguaje de modelado UML. Posteriormente se ha hecho una breve exposición de lo que puede aportar este lenguaje para el modelado de sistemas.

Basándose en esta metodología y este método de modelado se describirá en los siguientes capítulos el análisis y el diseño del sistema de gestión de la aplicación distribuida ABS, presentada en el capítulo anterior.

2.2. Análisis del sistema de gestión de ABS

2.2.1. Introducción

Este capítulo trata de analizar las funcionalidades más importantes a soportar por el sistema de gestión implementado para la segunda versión de la arquitectura del Servicio de Intermediación de ABS. Se enumerará y describirá las funcionalidades que ofrece el sistema de gestión, principalmente derivadas de aquellas descritas por la arquitectura de gestión de TINA-C. Más adelante se describirá una serie de casos de uso y diagramas de interacción para analizar el sistema y realizar la captura de requisitos. Estos diagramas han sido realizados mediante UML (*Unified Modelling Language*, Lenguaje de Modelado Unificado), lenguaje que se ha descrito en el capítulo anterior.

2.2.2. Funcionalidades de Gestión

El sistema de gestión del Servicio ABS es el encargado de controlar y monitorizar todos los recursos empleados en la provisión del servicio de intermediación en sí mismo. Las tareas de gestión se pueden dividir en cinco grupos de acuerdo con las áreas funcionales definidas en los estándares de gestión de OSI-SM: Fallos, configuración, contabilidad, rendimiento y seguridad (FCAPS, *Fault, Configuration, Accounting, Performance and Security*) [ITUT92a]. Esas áreas funcionales también han sido adoptadas por la arquitectura de gestión de TINA-C, arquitectura con la que trata de alinearse este sistema de gestión. La arquitectura de gestión de TINA-C describe las áreas funcionales que debieran aparecer en la gestión de los servicios desarrollados de acuerdo con la arquitectura de servicios de TINA-C (el servicio ABS es uno de dichos servicios):

1. **Gestión de contabilidad**: Esta funcionalidad se dedica a la "medida de servicios, derivando los cobros y la facturación de los subscriptores". Para realizar estas tareas, el sistema de gestión de ABS debería probablemente interactuar con la red, el sistema y los sistemas de gestión del *middleware* (para conocer el uso de los recursos subyacentes). Debería, así mismo, permitir el intercambio de información con los sistemas de gestión de otros proveedores de servicio de intermediación (para obtener una factura única en aquellos casos en que se use federación de intermediarios). Finalmente, debería existir la posibilidad, para los usuarios y/o proveedores de servicios y contenidos del servicio de

intermediación, de acceder a la información de contabilidad bajo demanda (la información de contabilidad mantenida por el servicio de gestión).

2. **Gestión de configuración:** De acuerdo con la arquitectura de gestión de TINA-C, esta área funcional se refiere a dos tipos principales de actividades: Gestión del ciclo de vida del servicio y del cliente.

   La gestión del ciclo de vida del servicio controla y monitoriza el "despliegue, operación y retirada" de los servicios. Está altamente relacionada con las actividades de gestión de configuración de *software* que están incluidas en los aspectos de gestión de la Arquitectura Computacional de TINA-C. En otras palabras, la gestión del ciclo de vida del servicio no se puede llevar a cabo sin una gestión de la aplicación de intermediación y del *middleware* (DPE, *Distributed Processing Environment* o Entorno de Procesamiento Distribuido). Por tanto, el sistema de gestión de ABS debería así mismo tener en cuenta estos aspectos de gestión (desde hace tiempo, la frontera entre la gestión de servicio y de aplicación no es muy clara [Pavlov97]).

   El otro conjunto de actividades, relacionadas éstas con la gestión del ciclo de vida del cliente, trata de la subscripción al servicio. Aunque la arquitectura de gestión de TINA-C no es muy clara en este punto, ejemplos de estas actividades podrían ser: Modificación de contratos de servicio, apunte o borrado de subscriptores por el administrador del servicio, etc.

3. **Gestión de fallos**: Esta área funcional incluye "actividades relacionadas con la detección, diagnosis y corrección de operaciones anormales" del servicio de intermediación. Esto implica la monitorización del rendimiento del servicio y los recursos subyacentes (red, sistemas de computación y entornos de procesamiento) para correlacionar fallos entre ellos. Conceptos como *trouble tickets* (recibos de problemas) y *fault log records* (registros de fallos) son de gran importancia en esta área.

4. **Gestión de Rendimiento**: Esta área funcional se refiere al "mantenimiento de los niveles de Calidad de Servicio (QoS, *Quality of Service*) acordados con los subscriptores para los servicios". La gestión de rendimiento, basada en la monitorización de distintos parámetros de rendimiento del servicio (número de peticiones servidas, tiempo medio de servicio, etc.), permitirán al proveedor del servicio saber cuándo el uso de recursos para proporcionar el servicio es correcto o no.

5. **Gestión de Seguridad**: El administrador del servicio debería poder, gracias al sistema de gestión, configurar varios aspectos relacionados con cuestiones de seguridad: Derechos de acceso, violaciones de seguridad, etc. Obviamente, esta gestión de seguridad dependerá

en gran medida de los particulares servicios externos e internos usados por el proveedor del servicio de intermediación.

Además de todas las funcionalidades que se tratará de soportar, mencionadas anteriormente, se deberán satisfacer los siguientes requisitos:

1. **Gestión Integrada**: La gestión del servicio se debería integrar con la gestión de la red, del sistema, del *middleware* y de la aplicación. En otras palabras, todos estos aspectos se deberían administrar usando el mismo marco de gestión (normalizado, a ser posible). De esta forma, se podría usar un único tipo de sistema de gestión. Los marcos normalizados de gestión son SNMP, OSI-MS (como base de la arquitectura de gestión de TMN) e incluso CORBA. Uno de estos marcos debe ser elegido para todos los casos de gestión.
2. **Control del servicio**: El sistema de gestión debería permitir a los administradores realizar un control directo sobre distintos aspectos de comportamiento del servicio.
3. **Gestión de Federación**: La arquitectura de ABS incluye la llamada federación de intermediarios. La federación de servicios se tiene en cuenta en la Arquitectura de Servicios de TINA y el punto clave en la gestión de este tipo de servicios es la gestión de contratos. Estos contratos además de información administrativa deberían contener:
    1. **Acuerdos de cobro**: Una indicación de cuanto debería pagar un proveedor de servicio a otro por el uso de sus servicios en un entorno federado.
    2. **Acuerdos de Nivel de Servicio** (SLA, *Service Level Agreements*): Estos SLAs contienen un conjunto de parámetros acordados de Calidad de Servicio que debería ser rellenado por los actores envueltos en la federación. Estos parámetros de QoS se podrían usar eventualmente por el sistema de gestión para realizar procedimientos de gestión de configuración para inicializar las aplicaciones que soportan el servicio de la forma en que se ha rellenado la QoS requerida y para derivar las políticas de monitorización necesarias para comprobar que los parámetros de SLA se están cumpliendo.

2.2.3. Casos de Uso

A continuación se muestran los casos de uso UML, explicados en la sección 2.1.3.3, que permitirán capturar los requisitos que deberá soportar el sistema de gestión de ABS. En ellos hay tres actores envueltos:

1. **Operador de pruebas**: Este actor modela a la persona responsable de monitorizar el estado del sistema de intermediación por medio de un conjunto de herramientas de gestión. Estas herramientas están fuera del sistema de intermediación.

2. **Aplicación de Intermediación**: Modela la aplicación de intermediación a gestionar. Se asume que la aplicación es distribuida y que está compuesta de un conjunto de componentes que interactúan. Estos componentes son los que mantienen la información en la que el operador de pruebas está interesado. La Aplicación de intermediación se podría considerar como otro paquete del sistema de gestión que se está modelando pero se prefiere modelar como otro actor para ocultar la forma concreta en que se instrumenta y accede a la información de gestión (un aspecto más relacionado con las etapas finales de diseño y la implementación).
3. **Sistema de Computación del Intermediario**: Este actor representa el sistema o sistemas de computación sobre el que corre la aplicación de intermediación. Ha sido modelada como un actor pues actúa (como en el caso de la Aplicación de Intermediación) como un proveedor de información de gestión.

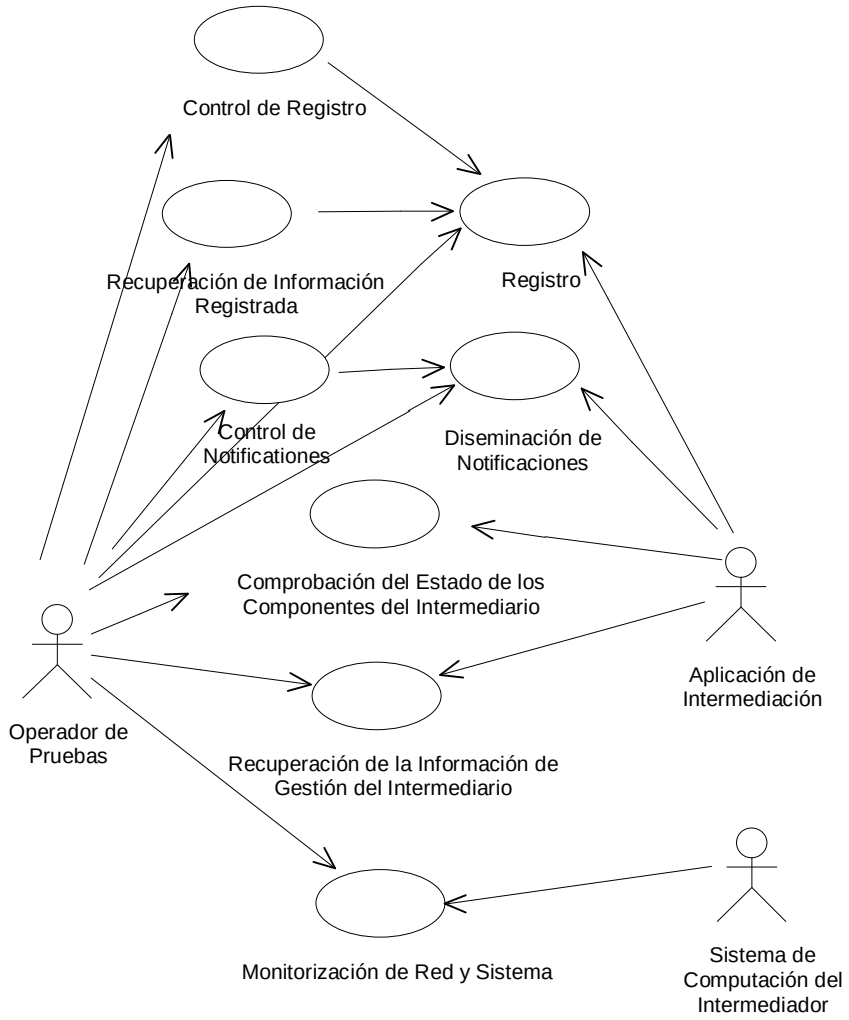


Figura 29. ***Casos de Uso para el Sistema de Gestión***

Con la definición de estos tres actores se puede enfocar el presente modelo en el sistema requerido para acceder a la información de gestión desde el exterior del sistema de intermediación. No se verá en este apartado la forma en que se obtiene la información del referido sistema.

Se han definido los siguientes casos de uso:

1. **Comprobación del estado de los componentes del Intermediario**: El operador de pruebas está interesado en obtener una vista general de qué componentes de la aplicación de gestión están lanzados y corriendo.
2. **Recuperación de la Información de Gestión del Intermediario**: Este caso trata de los pasos necesarios para recuperar los valores de las variables de gestión de los distintos componentes de la aplicación distribuida de intermediación.
3. **Monitorización de Red y Sistema**: El operador de pruebas (usando el mismo conjunto de herramientas que las empleadas para la gestión de la aplicación de intermediación) puede estar interesado en obtener alguna información sobre el rendimiento, configuración, etc., de los recursos de computación y comunicación de los sistemas subyacentes de computación.
4. **Diseminación de Notificaciones**: Los componentes de la aplicación de intermediación pueden generar notificaciones considerando circunstancias o condiciones especiales (por ejemplo, una actualización de la Red Conceptual). Estos informes o notificaciones se pueden reenviar al operador o se pueden almacenar para un acceso posterior.
5. **Control de Notificaciones**: El Operador debe poder especificar qué clase de notificaciones se le deben reenviar o deben ser almacenadas persistentemente.
6. **Registro**: Algunas notificaciones en concreto se pueden almacenar persistentemente para un acceso y procesado posterior.
7. **Recuperación de la Información Registrada**: El Operador debe poder acceder a la información registrada de notificaciones que previamente han sido realizadas por los componentes de la aplicación de Intermediación.
8. **Control de Registro**: El Operador debe poder controlar la forma en que la información registrada se almacena y procesa.

2.2.4. Jerarquía de Paquetes

La figura 30 muestra la jerarquía de paquetes del sistema de gestión de ABS. Como se comentaba en el apartado 2.1.3.2, los paquetes se utilizan para agrupar clases altamente relacionadas que son mutuamente cohesivas, y con un grado bajo de acoplamiento con otros

grupos. En este caso, el diagrama de paquetes presenta una primera descomposición funcional en la que se puede distinguir dos subsistemas:

1. **El Sistema Gestor**: Representa el conjunto de herramientas que usa el Operador de Pruebas para realizar sus tareas. Se compone de una Interfaz Gráfica de Usuario (GUI, *Graphical User* Interface) y un Paquete de Comunicaciones que estará a cargo de todos los detalles concernientes con el protocolo concreto de gestión estándar que se use (en este caso es SNMP).
2. **El Sistema de gestión del Intermediario**: Representa al conjunto de componentes del Dominio del Intermediario que permite el acceso a la información de gestión desde el Gestor del Sistema.

Se puede delimitar un conjunto preliminar de herramientas integradas bajo la GUI del Sistema Gestor:

1. **Visualizador de MIB (*Management Information Base*, Base de Información de Gestión)** para navegar por el conjunto de variables u objetos de gestión definidos.
2. **Herramienta de Monitorización Gráfica** que recupere periódicamente el valor de una variable de gestión y la visualice.
3. **Herramienta de Comprobación de Componentes de la Aplicación** que represente gráficamente el estado de los Componentes de la Aplicación de Intermediación.

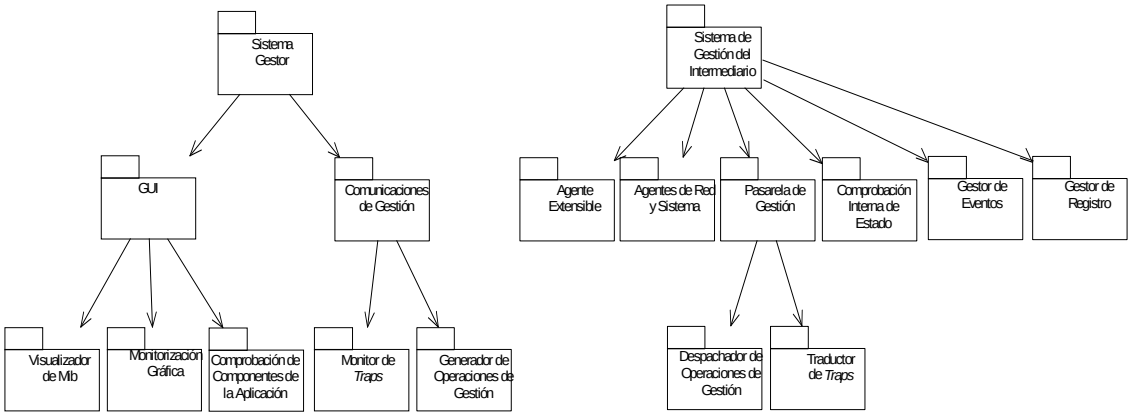


Figura 30. ***Jerarquía de Paquetes***

El Paquete de Comunicaciones se divide en:

1. **Monitor de *Traps***: Se encarga de recibir la información de gestión generada por el sistema gestionado (la aplicación de intermediación o su sistema soportado) sin una petición previa del operador.

2. **Generador de Operaciones de Gestión**: Envía peticiones de operaciones de gestión (de acuerdo con el protocolo de gestión elegido, en este caso SNMP) al sistema gestionado.

Con respecto al Sistema de Gestión del Intermediario, se identifican los siguientes paquetes:

1. **Agente Extensible**: Este componente recibe las operaciones de gestión del Generador de Operaciones de Gestión y decide si la petición corresponde a información de gestión mantenida por la aplicación de intermediación o concierne a información general de sistema o red. Entonces la reenvía al componente apropiado (la pasarela de gestión de los agentes de Red y Sistema).
2. **Agentes de Red y Sistema**: Este paquete contiene todos los agentes instalados en el sistema de computación que soporta la aplicación de gestión. Estos agentes estarán principalmente a cargo de la gestión genérica de red y sistema. No son específicos del entorno del intermediario (más aún, se trata de reutilizar agentes existentes).
3. **Pasarela de Gestión**: Este paquete contiene todos los componentes necesarios para recibir operaciones de gestión del Agente Extensible y redirigirlas al componente de la aplicación de intermediación adecuado (Despachador de Operaciones de Gestión). También se encarga de la Traducción de *Traps*. Así mismo realizará la conversión de operaciones entre distintos formatos y protocolos (SNMP/CORBA en este caso).
4. **Comprobación Interna de Estado**: Este paquete se encarga de recoger información sobre el estado de la aplicación de intermediación para informar asíncronamente ante un cambio al sistema gestor.
5. **Gestor del Registro**: Este paquete agrupa el conjunto de clases que se necesitan para recibir y almacenar diferentes tipos de notificaciones realizadas.
6. **Gestor de Notificaciones**: Este paquete se encarga de la recepción de notificaciones realizadas por los componentes de la aplicación de Intermediación y de su reenvío al Gestor del Registro, el Traductor de *Traps* o ambos.

2.2.5. Diagramas de Interacción

Los siguientes diagramas de interacción de alto nivel muestran el diálogo entre los actores y las unidades funcionales identificadas en los diagramas de casos de uso y de paquetes anteriormente descritos.

2.2.5.1. Diagrama de Interacción de Comprobación del Estado de los Componentes del Intermediario

Tal y como se muestra en el diagrama de interacción de la figura 31, el paquete de Comprobación Interna de Estado contiene todas las clases necesarias para realizar una comprobación periódica de los componentes de la Aplicación de Intermediación. De acuerdo con la información obtenida, envía información de configuración filtrada (si un componente está funcionando o caído) al sistema gestor a través del Traductor de *Traps*, el cual realiza las conversiones de formatos y protocolos necesarias. El Demonio de *Traps* en el sistema gestor recibe la información (si se ha configurado previamente para ello, ver pasos 1 y 3) y la pasa a la GUI para una presentación adecuada.

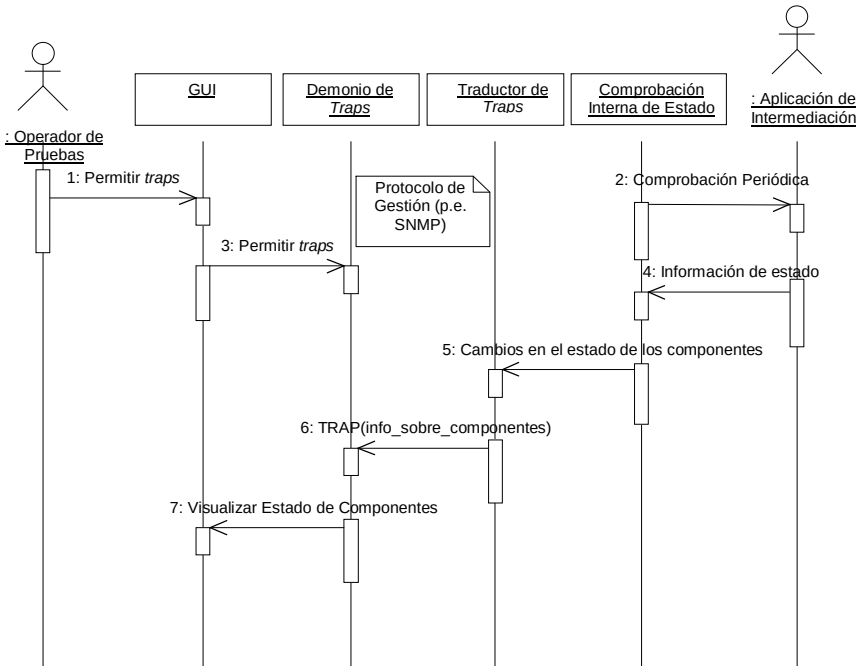


Figura 31. Comprobación del estado de los componentes del Intermediario

2.2.5.2. Obtención de Información de Gestión del Intermediario

El Usuario de Pruebas selecciona la parte de la información de gestión que quiere recuperar (puede corresponder indistintamente a la aplicación, la red o el sistema de gestión) en la GUI (pasos 1, 2 y 3 de la figura 32). La GUI pasa la petición al Subsistema de Comunicaciones de Gestión y más concretamente al llamado Generador de Operaciones de Gestión, el cual envía una operación de gestión al sistema gestionado de acuerdo con el protocolo de gestión elegido (SNMP en este caso). Dicha petición la recibe el Agente Extensible quien decide quién puede completarla: La Aplicación de Intermediación o el

sistema de computación subyacente (si la petición se refiere a la gestión de la red o el sistema). Si la petición se refiere a la Aplicación de Intermediación, la Pasarela de Gestión traduce la petición en una operación entendible por dicha Aplicación. En este caso, el Despachador de Operaciones de Gestión realiza la traducción y, a la vez, decide qué componente puede completar la petición. Solicita la información de un componente en concreto y la información obtenida (o un error, que no se muestra en la figura 32) sigue el camino inverso hacia el Sistema Gestor.

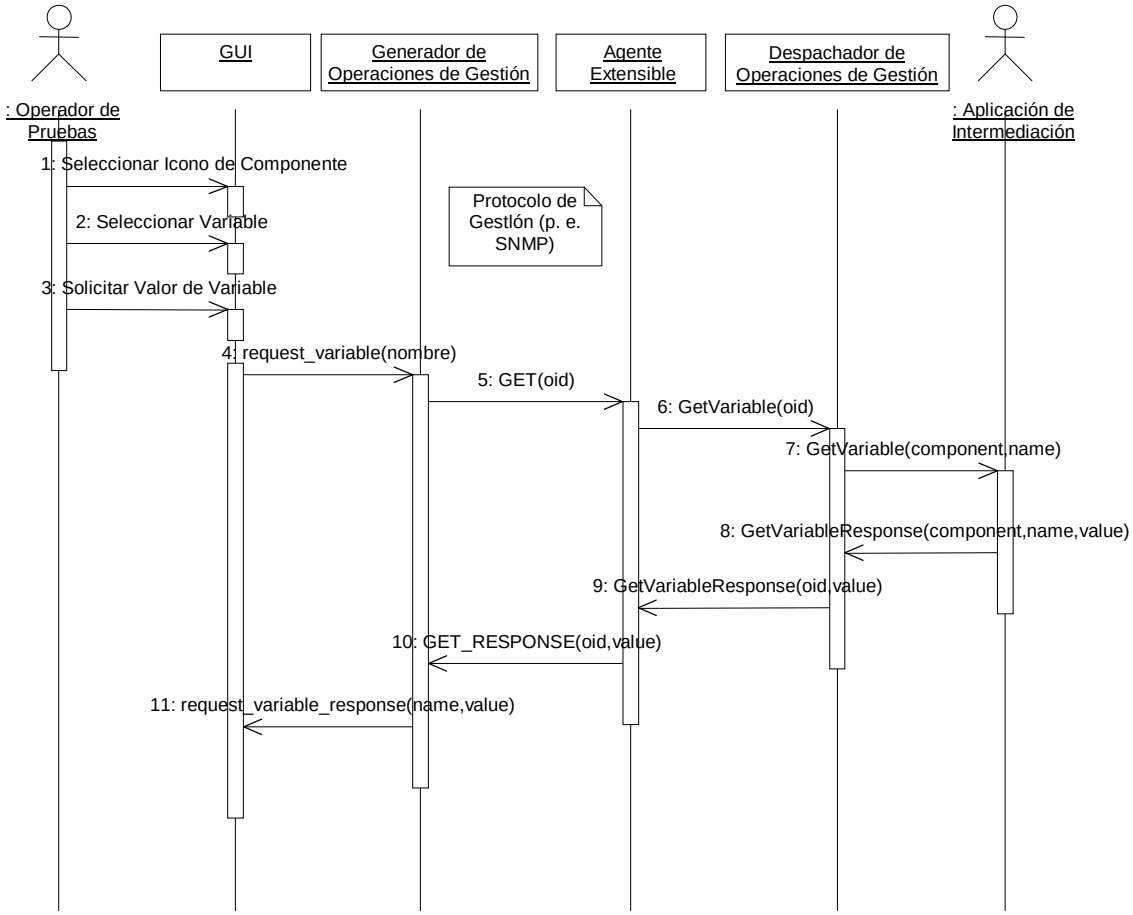


Figura 32. ***Recuperación de información de gestión del Intermediario***

2.2.5.3. Monitorización de Red y Sistema

El diagrama de Interacción mostrado en la figura 33 es similar a la Recuperación de Información de Gestión del Intermediario, pero en vez de reenviar la petición al Repartidor de Operaciones de Gestión, el Agente Extensible la reenvía al Agente de Red o Sistema apropiado (posiblemente sin realizar traducciones de tipo alguno), el cual, al mismo tiempo, obtiene la información del llamado Sistema de Computación del Intermediario (ordenador(es) donde la Aplicación de Intermediación corre).

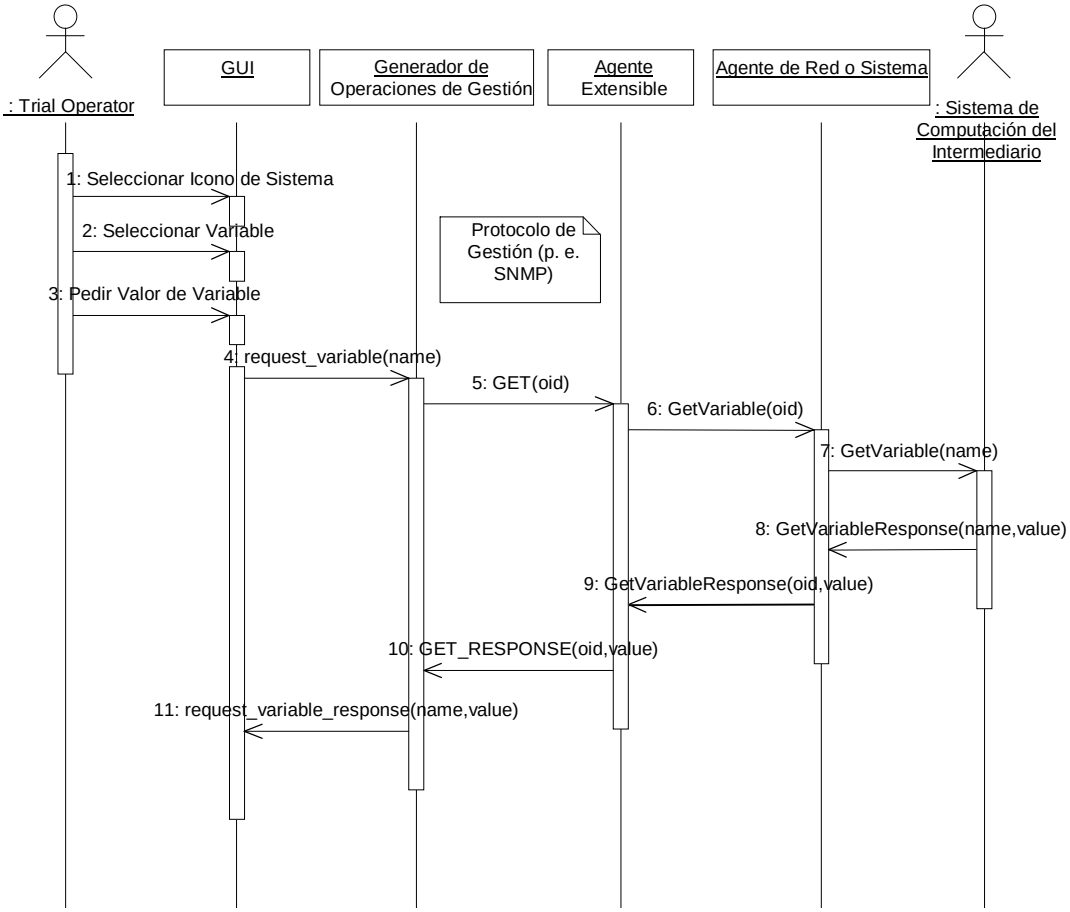


Figura 33. ***Monitorización de red y sistema.***

2.2.5.4. Diseminación y Registro de Notificaciones

Los Componentes de la aplicación de intermediación emiten notificaciones enviándolas al Gestor de Notificaciones, el cual, de acuerdo a su configuración, decide si una notificación en particular se debe enviar al Gestor del Registro, al Operador del Sistema Gestor o a ambos. La figura 34 muestra este diagrama de interacción.

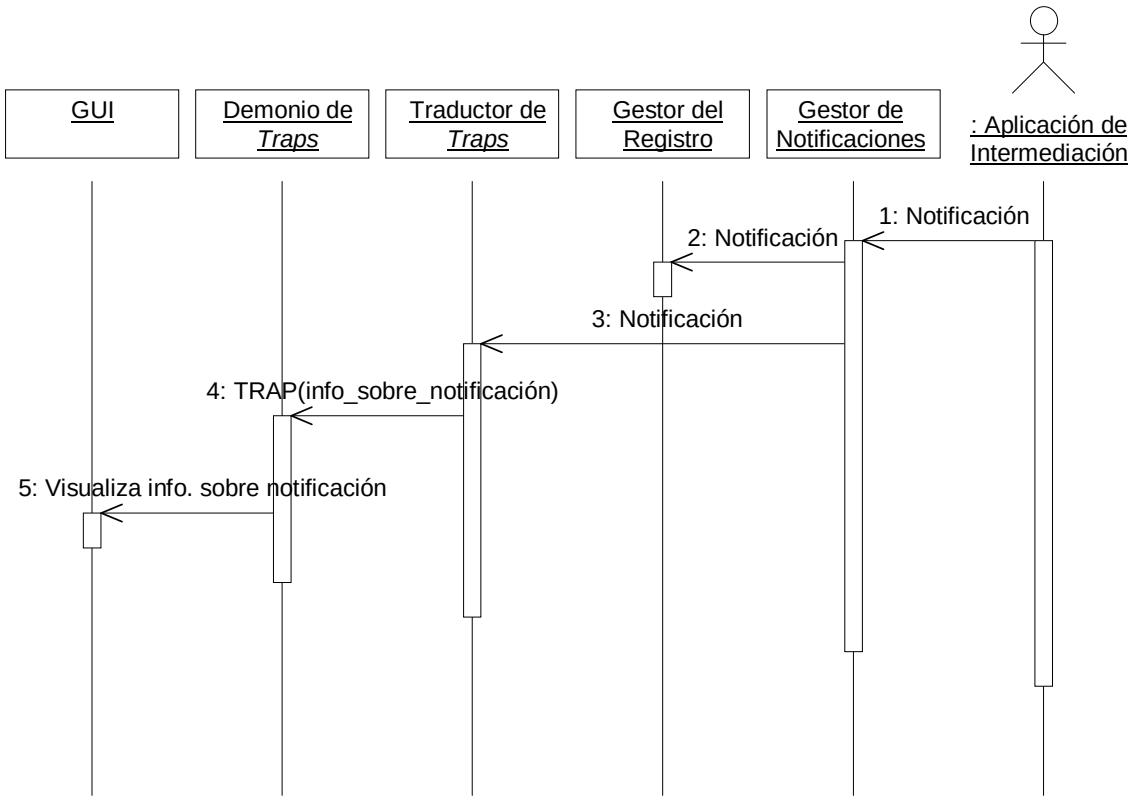


Figura 34. ***Diseminación de notificaciones***

2.2.5.5. Recuperación de Información del Registro

Este caso se puede considerar una particularización del caso de uso de la Recuperación de Información de Gestión del Intermediario si se considera al Gestor del Registro como otro componente de la Aplicación de Intermediación. La figura 35 muestra este diagrama de interacción. La información del registro se trata como información de gestión genérica.

Otra posibilidad a tener en cuenta sería que el Gestor de Registro pudiera eventualmente configurarse para emitir notificaciones bajo circunstancias especiales recordando la información registrada (por ejemplo muchos fallos al registrar accesos). Estos eventos serían enviados al Gestor de Notificaciones de la misma forma que el diagrama de interacción de la figura anterior.

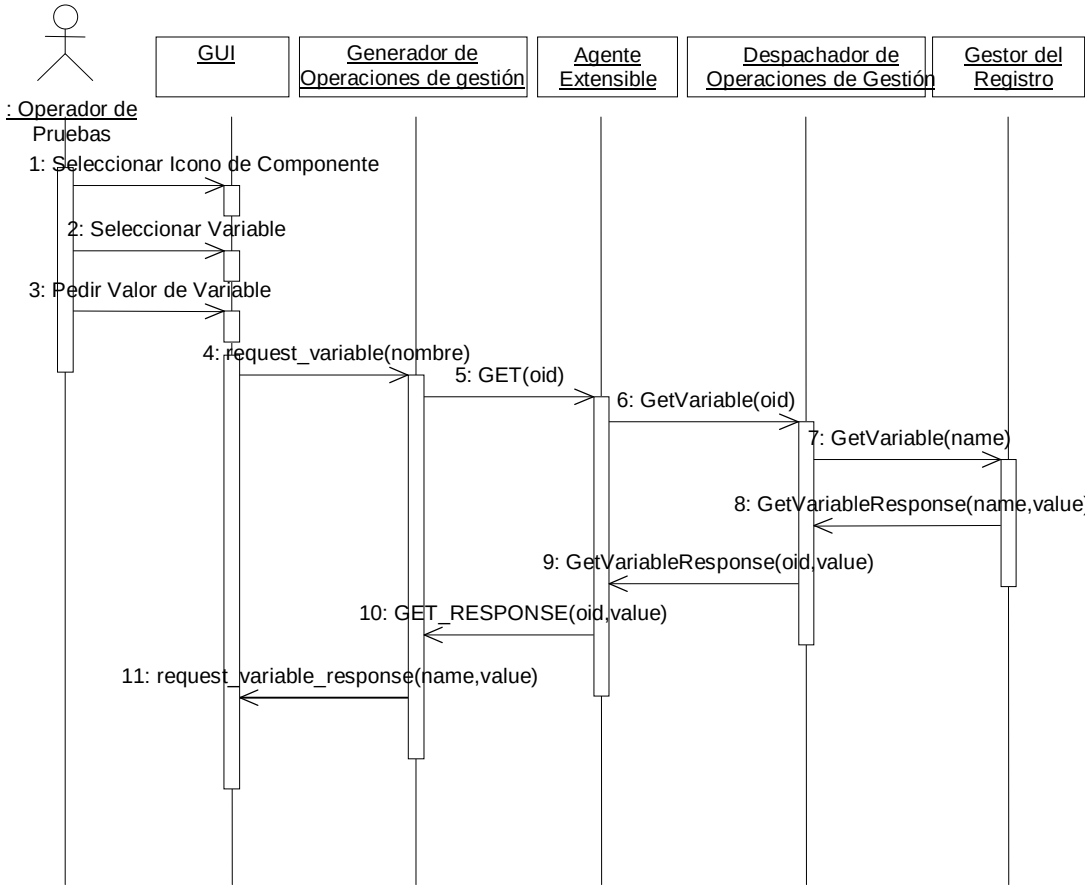


Figura 35. ***Recuperación de información del registro***

2.2.5.6. Control de Registro y Notificaciones

En estos casos, el Operador del Intermediario, a través del sistema gestor, configura el comportamiento del Gestor de Registro y el de Notificaciones. La figura 36 muestra el diagrama de interacción para el caso de uso del Control de Notificaciones. El Control de Registro es similar (cambiando el Gestor de Notificaciones por el del Registro). Por medio de estas operaciones de control, el Operador podría, por ejemplo, definir qué tipo de notificaciones se deben reenviar al Sistema Gestor.

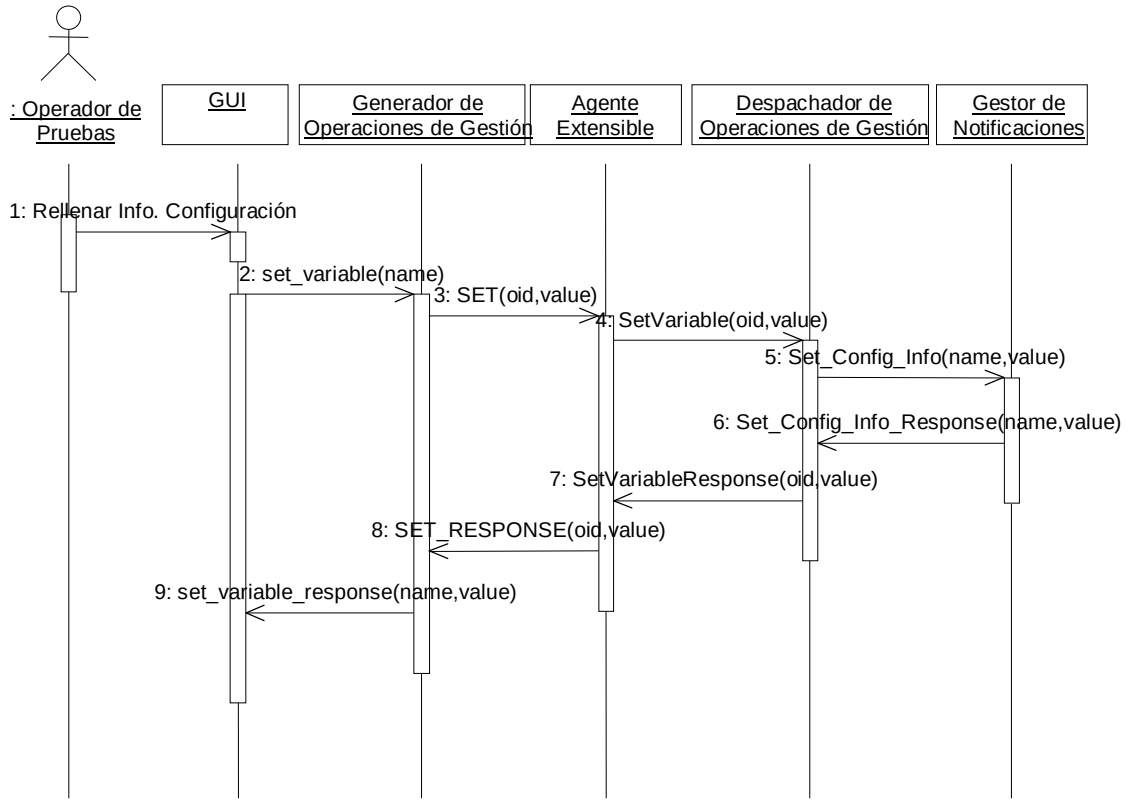


Figura 36. ***Control de Notificaciones***

2.2.6. Definición de objetos de información

Los diagramas de interacción anteriores mostraban múltiples intercambios de información de gestión entre los actores y paquetes anteriormente descritos. En esta sección se define toda esta información intercambiada, de la cual depende la capacidad de cada uno de los componentes de la aplicación de intermediación de ser gestionado.

2.2.6.1. Definición de Información de Gestión

Un aspecto clave en el Sistema de Gestión del Intermediario es la definición de la información de gestión, especialmente la relacionada con la gestión de la aplicación de intermediación. A continuación se enumera un conjunto de variables de gestión propuesto para la monitorización de dicha aplicación.

- Número de usuarios registrados.
- Número de proveedores de contenido registrados.
- Número total de sesiones usuario-intermediario.
- Número total de sesiones intermediario-proveedor de contenidos.
- Número de sesiones usuario-intermediario actuales.

- Número de sesiones intermediario-proveedores de contenido actuales.
- Número de demandas de usuarios.
- Demandas de usuario encaminadas al QPG.
- Demandas de usuario encaminadas al NE
- Número de operaciones de navegación fallidas.
- Número de consultas no resueltas debido a la falta de información.
- Número de consultas de usuario refinadas.
- Número de consultas resueltas con información local.
- Número de consultas resueltas con acceso a los CP.
- Número de consultas resueltas por agentes de tipo x.
- Número de operaciones de agentes de acceso fallidas debido a problemas de comunicación.
- Número de recursos almacenados en el RM.
- Número de artículos almacenados en el LRM (*Local Resource Manager*, Gestor de Recursos Locales)
- Número de actualizaciones de la base de datos local desencadenadas.
- Número de respuestas procesadas por el RA (*Resource Agent*, Agente de Recursos).
- Número de respuestas no cambiadas por el RA.
- Número de conceptos en el CNM.
- Número de perspectivas en el CNM.
- Número de puentes en el CNM.
- Número de objetos proyectados en el CNM.
- Número de consultas resueltas en cada perspectiva.
- Número de puentes usados.

Así mismo, se mantiene la información definida para la primera versión de la aplicación de intermediación.

- *qpgMostfreqViewpoint*: El nombre de la perspectiva usada con más frecuencia.
- *qpgMostfreqSO*: El nombre del Objeto Proyectado solicitado con más frecuencia.
- *qpgNumofreceivedQuery*: Número de consultas recibidas.
- *qpgNumofUnresolvedqueries*: Número de consultas sin resolver debido a una pérdida de conexión.
- *qpgNumofPlans*: Número de planes generados.
- *qpgPlanGenerationTime*: Tiempo de procesado en la generación del último plan.
- *rmNumofNewOffers*: Número de nuevas ofertas en el RM

- *rmNumofModifiedOffers*: Número de ofertas modificadas en el RM.
- *rmNumofWithdrawOffers*: Número de ofertas borradas.
- *rmNumofRegisteredCP*: Número de proveedores de contenido registrados en el RM
- *qeeAvPlanTime*: Tiempo medio consumido en la ejecución de un Plan.
- *qeeAvSubGoalTime*: Tiempo medio consumido en la ejecución de una Sub-meta.
- *neNumNavigationRequests*: Número de peticiones de navegación procesadas por el NE.
- *neNumNavigationFailures*: Número de peticiones de navegación que el NE no ha podido procesar.
- *neDepthLevel*: Profundidad de una petición de navegación (parámetro de lectura y escritura).
- *oamgrMostUsedCP*: El nombre del proveedor de contenido usado con más frecuencia.
- *oamgrNumofConnections*: Actual número de conexiones establecidas con el OAM.
- *uamAvQueryTime*: Duración media de una consulta.
- *uamAvNavigTime*: Duración media de una operación de navegación.

En cualquier caso, el formato de estos datos dependerá del protocolo de gestión elegido. Si se usa SNMP, como es el caso, habrá que definir una MIB de ABS en SMI.

2.2.6.2. Notificaciones

El formato de las notificaciones que usarán los Componentes de la Aplicación de Intermediación es un aspecto clave del modelo de información del Sistema de Gestión. Está basado en el formato de la estructura de datos `StructuredEvent` definida en la actual propuesta de un Servicio de Notificaciones que está estudiando OMG [OMG98a]. Se ha elegido esta estructura debido a su flexibilidad y evita varios pasos de encapsulación y extracción (que serían necesarios en el caso de usar Eventos no Tipados definidos en el Servicio de Eventos de OMG [OMG97e]. En este caso, los eventos se definirían simplemente como tipos `Any`).

La estructura propuesta, en IDL, se muestra en el cuadro siguiente:

```
struct Property   {
      string name; // Nombre de la propiedad
      any value; // Valor encapsulado en un Any
};

typedef sequence<Property> PropertySeq;

typedef PropertySeq FilterableEventBody;
```

```
struct EventHeader { // Cabecera de la notificación
      string event_type;
      string event_name;
      // La especificación de OMG ha definido algunos campos
      // opcionales, p. e. EventReliability,
      // ConnectionReliability, Priority, StartTime, StopTime
      // TimeouT, pero no se han tenido en cuenta.
};

struct StructuredEvent  {
      EventHeader header;
      FilterableEventBody filterable_data;
};
```

Cuadro 11. ***Definición de las notificaciones***

En este caso, la notificación se puede descomponer en:

1. Una cabecera, con nombre y tipo de evento.
2. Un cuerpo de la notificación, con un conjunto de pares nombre-valor que puede estar sujeto a filtrado (de acuerdo a las propiedades de configuración del Gestor de Notificaciones). Estas parejas consisten en una cadena de caracteres para el nombre y un `Any` para el valor.

2.2.7. Conclusiones

Este capítulo ha tratado de realizar un análisis del sistema de gestión a desarrollar. Se ha enumerado y descrito las funcionalidades que debe ofrecer el sistema de gestión, principalmente derivadas de aquellas descritas por la arquitectura de gestión de TINA-C. Para ello se ha utilizado una serie de casos de uso y diagramas de interacción.

Una vez se ha concluido esta fase del ciclo de desarrollo, se puede pasar a la fase de diseño, en la que se tratará de explicar qué métodos se han empleado para traducir los datos obtenidos en la fase de análisis en otros más concretos. Con dichos datos, finalmente se realizará la codificación y pruebas del sistema desarrollado.

2.3. Diseño del sistema de gestión de ABS

2.3.1. Introducción

El presente capítulo pretende dar una visión detallada de las características de diseño del sistema, una vez se ha mostrado el análisis del sistema a implementar en el capítulo anterior. Trata de utilizar los conocimientos mostrados en los capítulos preliminares, que introducían temas de gestión de red, procesamiento distribuido, gestión de aplicaciones distribuidas y de intermediación electrónica, integrando la gestión de una aplicación CORBA en un entorno de gestión tradicional SNMP, para aplicarlos al diseño del sistema que se ha implementado finalmente.

Para explicar algunas cuestiones de diseño ha sido necesario hacer referencia al código implementado. Estas referencias no llegan a un gran nivel de profundidad, pero es importante recordar que la parte del sistema que se ejecuta en el dominio CORBA se ha implementado en Java. Esto implica que el lector deberá conocer los algoritmos que se utilizan a la hora de trabajar e implementar objetos CORBA con este lenguaje. En el segundo capítulo se puede encontrar información sobre este tema.

El capítulo se ha dividido en varias partes que son, aparte de esta introducción:

1. **Requisitos y restricciones**: En este apartado se trata de explicar los requisitos y restricciones que llevaron a tomar las decisiones de diseño utilizadas finalmente.
2. **Arquitectura**: Este apartado muestra esquemáticamente, pues en siguientes apartados se desarrolla, el diseño global de la arquitectura del sistema de gestión.
3. **Modelo de información**: Este apartado intenta mostrar cual ha sido el modelo de información empleado en el diseño.
4. **Pasarela SNMP-CORBA**: Este apartado trata una parte importante del sistema de gestión y es aquella que se encarga de pasar la información de uno a otro dominio.
5. **Interfaces CORBA de gestión**: Este apartado muestra cómo son las interfaces utilizados por los componentes para instrumentar su gestión.
6. **Consola de Gestión**: Este apartado incluye toda aquella información relacionada con el diseño de la Consola de Gestión que utilizará el Operador del sistema de Intermediación.

Finalmente, acaba el capítulo con otro apartado adicional de conclusiones mostrando, de forma resumida cómo se ha conseguido una integración entre la gestión de red y la de aplicación.

2.3.2. Requisitos y restricciones

A la hora de diseñar la arquitectura de gestión de la aplicación de intermediación hubo que tomar una serie de decisiones. Muchas de ellas fueron en parte motivadas por el diseño del sistema global de Intermediación de ABS, y se pueden encontrar en [ABS98]. Estas se indican a continuación:

1. El sistema de gestión no tenía que ser implementado desde cero. Se debía reutilizar la mayor cantidad de componentes existentes de otros sistemas de gestión cuando fuera posible. Por ejemplo, no se debía crear una aplicación que actuara como consola de gestión para este sistema particular, sino que sería mejor adaptar una de las que ya existiera en el mercado.

2. El operador del sistema debía ser capaz de obtener información de todas las entidades que tuvieran implicaciones en ABS de forma conjunta. Esto suponía que la gestión de la aplicación se tenía que integrar con aspectos de la gestión de la red y el sistema.

3. El desarrollo de los diferentes componentes de la aplicación debía verse afectado lo menos posible por la funcionalidad de gestión. Esto redundaría en una mayor productividad, al separar claramente las tareas de cada desarrollador.

Estos puntos llevaron a decisiones de diseño que fueron tomadas durante las primeras etapas del desarrollo del sistema de gestión de ABS:

1. El sistema de gestión debería estar basado en el estándar SNMP [Case90]. La simplicidad del marco de gestión de SNMP, el gran número de plataformas de gestión SNMP disponibles (incluso de dominio público), y la disponibilidad de agentes SNMP de red y sistema fueron factores clave para esta elección. También se consideró el uso del marco de gestión OSI, dada su potencia y los esfuerzos realizados en el campo de la gestión CORBA/CMIP [Open97][OMG98b]. Pero dicha potencia implicaba una complejidad que no era necesaria para las características del sistema de gestión de ABS.

2. Se adoptó como aproximación una pasarela SNMP-CORBA para alcanzar interoperabilidad entre los marcos SNMP y CORBA [OMG98c]. Se podría haber elegido la posibilidad de un gestor *multiarquitectural*, tal y como describe en [Keller98]. En dicho artículo se propone una plataforma que soporta un conjunto de protocolos de gestión que están implementados en la pila de comunicación de dicha plataforma. Por tanto, la conversión entre distintos protocolos (SNMP y CORBA) mediante una pasarela no es

necesaria. Sin embargo, se prefirió hacer el sistema de gestión de ABS tan independiente como fuera posible de los servicios de una plataforma particular de gestión SNMP, con lo que se prefirió rechazar la posibilidad antes mencionada.

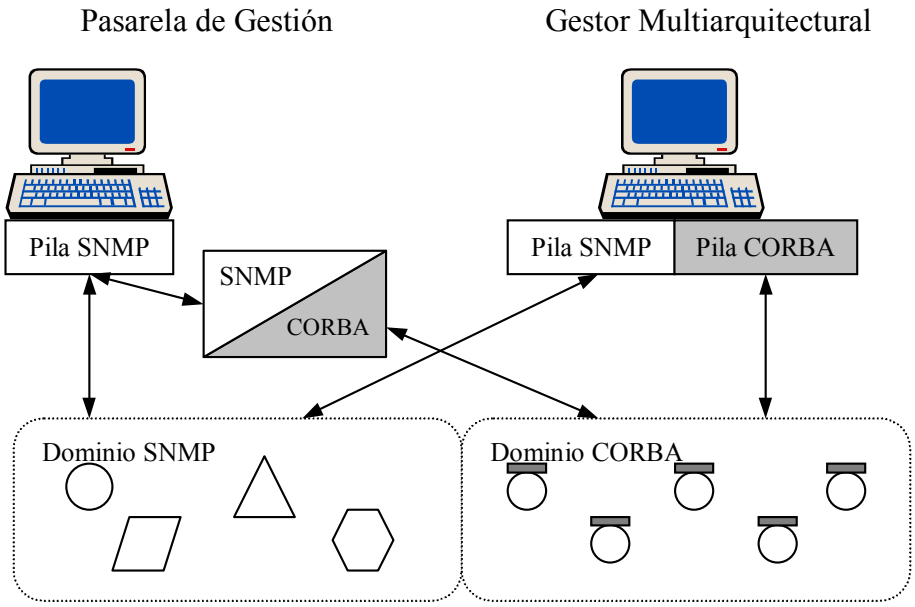


Figura 37. ***Posibles aproximaciones a la gestión integrada de red y aplicaciones***

En la siguiente sección, se describirá en detalle la arquitectura de gestión de ABS, tratando de explicar cómo ha influenciado en su diseño los diferentes aspectos de otras iniciativas sobre gestión de aplicaciones basadas en CORBA, tales como el uso de los algoritmos de traducción desarrollados por JIDM (*Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios), ya descrito en un capítulo anterior.

2.3.3. Arquitectura del sistema de gestión de ABS

La presente figura ya fue mostrada en el capítulo que se dedicaba a la presentación del proyecto ABS (figura 24). Muestra la arquitectura global de la aplicación de intermediación. Sin embargo, en este caso cambia la perspectiva desde que se enfoca. En la figura 38, en estudio, se muestra la perspectiva de gestión, y así, se ve como el componente BSM (*Broker System Manager*, Gestor del Sistema de Intermediación) interactúa con el resto de los que están en el dominio del intermediario (flechas grises). El BSM es el componente que se encarga de la gestión del sistema en el dominio del Intermediario y ha sido desarrollado en su totalidad en este Proyecto Fin de Carrera. Por otra parte, la OT (*Operator Terminal*, Consola

del Operador), en el dominio del Operador, se comunica con el BSM por medio del MSA (*Management Service Agent*, Agente para la Gestión del Servicio).

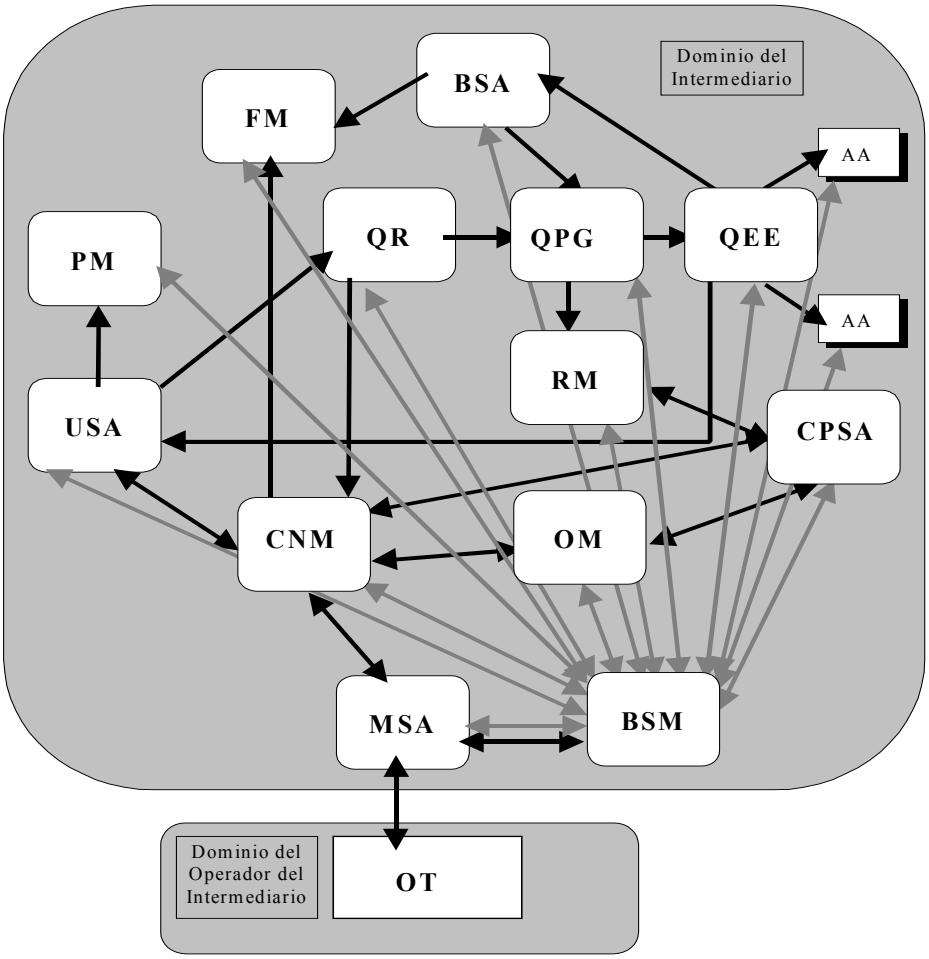


Figura 38. ***Bloques funcionales de la arquitectura del servicio de intermediación de ABS. Perspectiva de gestión***

A continuación se muestra en otra figura la arquitectura del sistema de gestión de ABS desarrollado para la segunda versión de la arquitectura del servicio de intermediación de ABS.

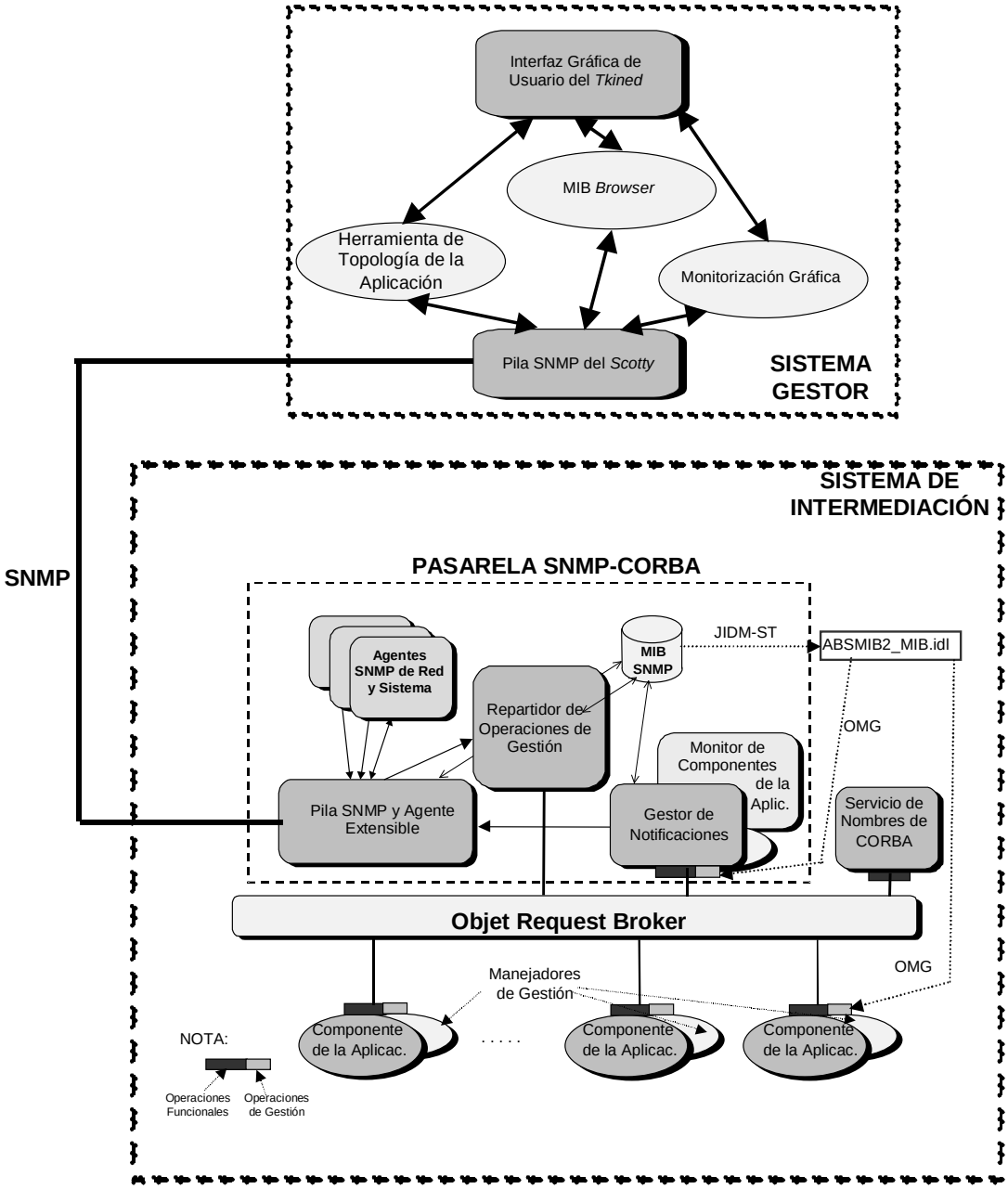


Figura 39. ***Arquitectura del sistema de gestión de ABS***

Al realizar un estudio comparativo, se puede establecer una relación entre la primera y la segunda figura. El OT es el nombrado gestor del sistema. El MSA es aquella parte de la arquitectura que realiza la función de interfaz entre la OT y el Sistema de Intermediación, y es, por tanto, el bloque compuesto por la pila SNMP y el Agente Extensible. Lo que resta de la pasarela CORBA/SNMP es el BSM. Una vez establecidas las relaciones pertinentes, las diferentes partes de la arquitectura del sistema de gestión se describirán en las siguientes secciones.

Por último se muestra un par de figuras más, con sendos diagramas de clases que muestran la estructura diseñada para la consola de gestión y el BSM respectivamente. El contenido de dichos diagramas se tratará de desarrollar a lo largo del resto de capítulo.

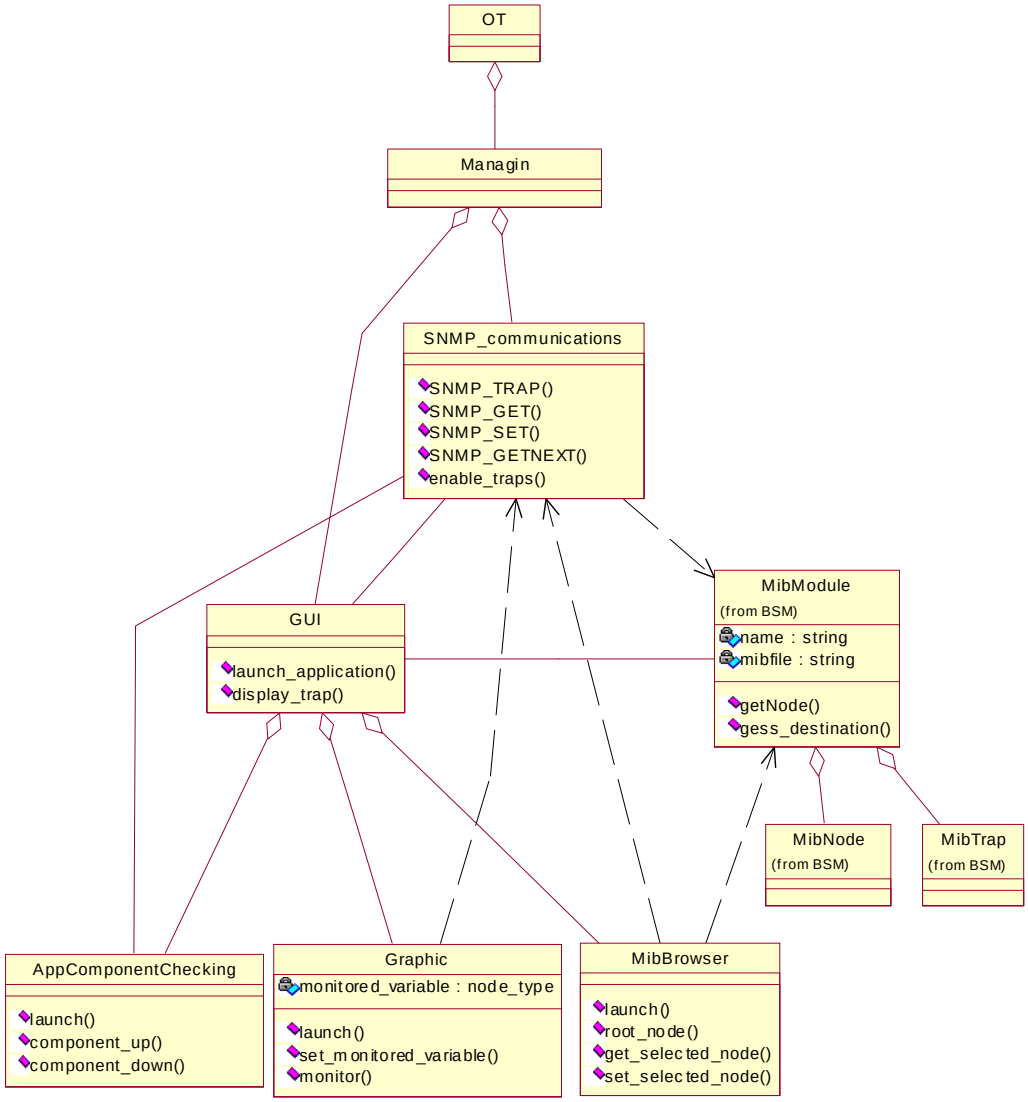


Figura 40. ***Diagrama de clases del OT-MGR.***

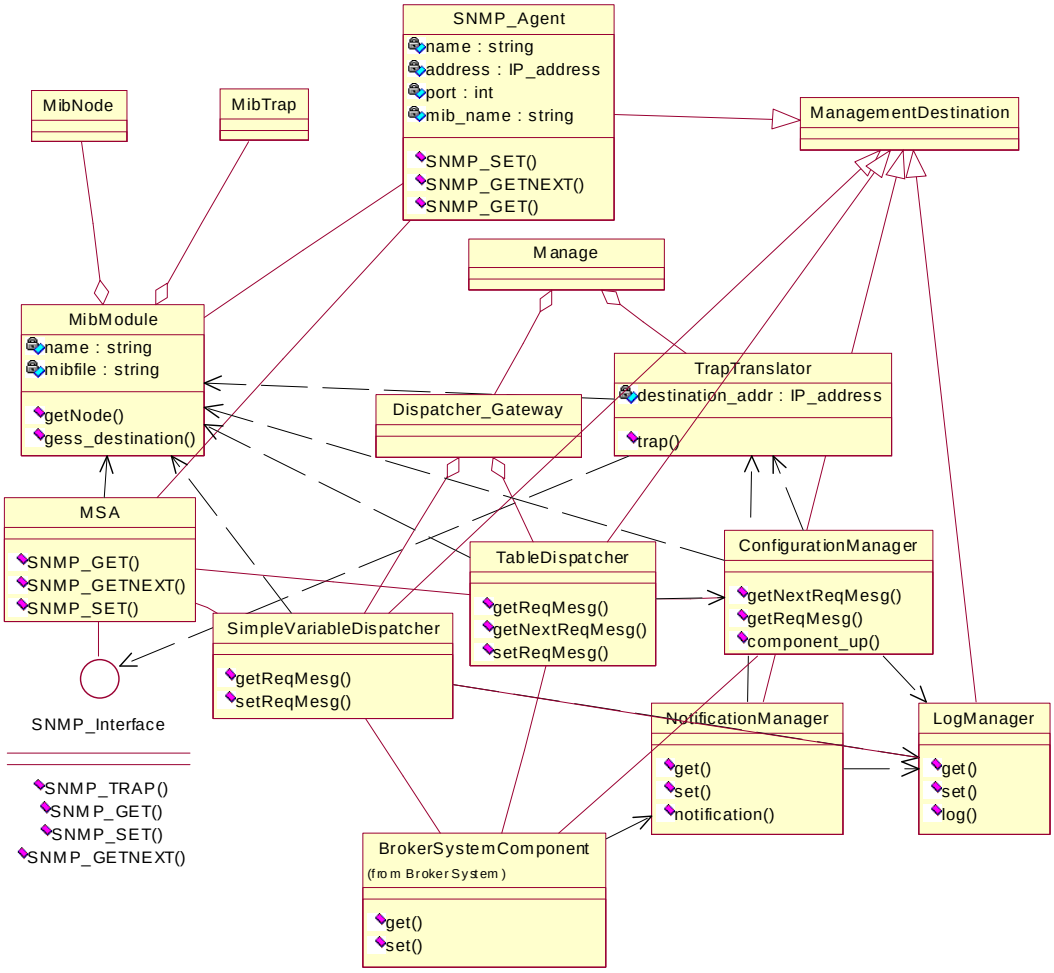


Figura 41. ***Diagrama de clases del MSA y el BSM***

2.3.4. Modelo de información de gestión.

Este apartado intenta mostrar cuál ha sido el modelo de información empleado en el diseño. Se ha definido la información de gestión específica de la aplicación y el servicio de ABS de acuerdo con las reglas de la SMI (*Structure of Management Information*, Estructura de la Información de Gestión) de Internet [Rose90]. La especificación definida se traducirá luego a IDL utilizando los algoritmos descritos en la Traducción de Especificaciones de JIDM [Open97], estudiados en el segundo capítulo.

El uso de la SMI de Internet para la definición de la información de gestión en este tipo de escenarios (gestión SNMP de una aplicación CORBA) se sugiere también en [OMG98c]. Esta cita forma parte de la Traducción de Interacciones de JIDM y en ella se muestra los pasos a realizar si se está interesado en soportar la gestión de objetos CORBA desde un gestor basado en SNMP o en CORBA y la implementación del objeto CORBA está

todavía por hacer. En este caso, en el momento del diseño era posible tomar esta solución, pues se aproximaba en gran medida a lo que se necesitaba.

La MIB de ABS desarrollada define un grupo SMI para cada componente de la aplicación de ABS. En estos grupos hay información de gestión relacionada con aspectos del servicio de intermediación y también hay definiciones de variables de gestión para controlar la parte de gestión del componente correspondiente.

Por ejemplo, la figura 42 muestra el grupo SMI *qgp* que contiene la información de gestión mantenida por el componente QPG (*Query Plan Generator*, Generador de Plan de Consultas) del sistema de intermediación de ABS. La variable SMI `qpgTimeBetweenNotifications` permite al sistema de gestión controlar la forma en que el componente QPG debe comportarse con respecto al sistema de gestión. Las otras variables contienen información sobre el servicio proporcionado por el QPG. La MIB creada para ABS se puede consultar en la sección de planos de este proyecto.

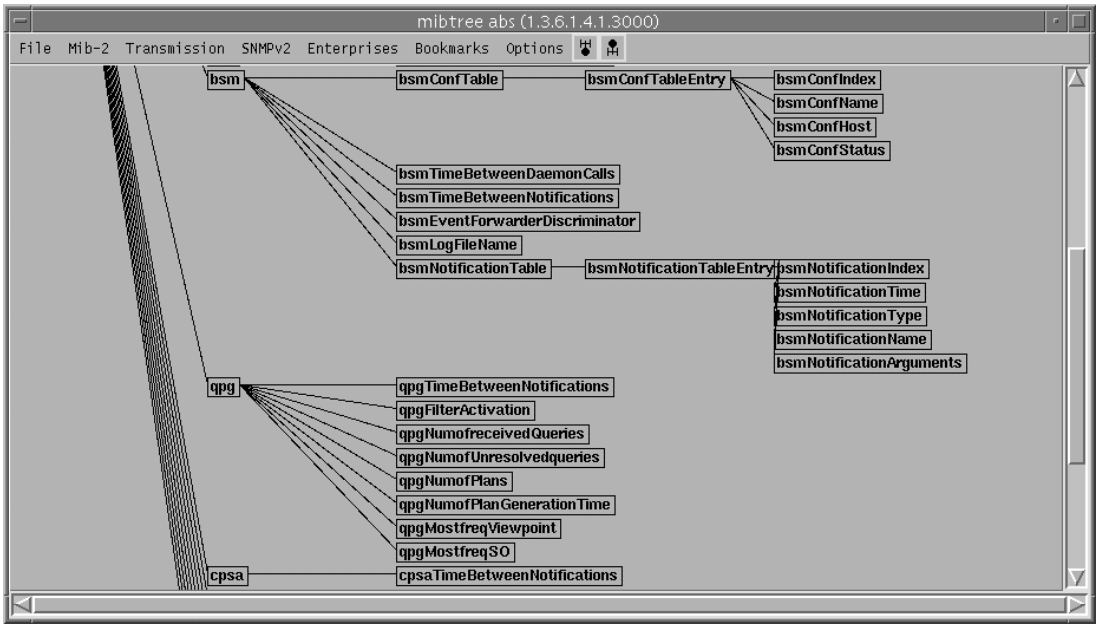


Figura 42. ***Parte de la MIB de ABS***

La información de gestión es mantenida en parte por la aplicación y en parte el propio BSM para configurar cómo se llevarán a cabo sus operaciones. En cualquier caso, es transparente al gestor quien mantenga esta información de gestión. Más adelante se verá qué información es específica al BSM.

Así mismo se propuso una serie de *traps* SNMP para traducir aquellas notificaciones de máxima prioridad a SNMP: Activación y caída de componentes y operaciones de gestión irregulares.

Para la gestión de red y sistema se hizo uso de las definiciones existentes de información de gestión. En concreto, la MIB-II de Internet [McCloghrie91b] y la MIB de *Sun* [SunSoft94], dado que el sistema funciona sobre una máquina de este vendedor.

2.3.4.1. Información del Gestor del Sistema de Intermediación.

Merece una mención aparte la información que maneja el BSM (*Broker System Manager*, Gestor del Sistema de Intermediación). Como ya se ha comentado anteriormente, este componente de la aplicación de intermediación es el encargado de realizar las funciones de gestión en el dominio CORBA Realiza varias de las funcionalidades mostradas en los casos de uso del capítulo de análisis, entre las que se encuentra la monitorización del sistema y la gestión de notificaciones y registros. Es por ello por lo que hay que prestarle una mayor atención.

Las variables de gestión definidas para este componente han sido definidas para controlar desde la Consola del operador su comportamiento. Esto permite variar los intervalos entre monitorizaciones o el modo en que se manipula las notificaciones. Así, nos encontramos con las siguientes variables:

1. `bsmTimeBetweenNotifications`: Controla el intervalo en segundos entre envío de notificaciones periódicas indicando al sistema gestor que el BSM está corriendo. Esto permite asegurar al Operador que el estado del sistema que se visualiza en la GUI es correcto. Se puede desactivar el envío de notificaciones poniendo esta variable a cero.

2. `bsmTimeBetweenDaemonCalls`: Controla el intervalo en segundos entre peticiones al demonio de Orbix de los servidores que hay corriendo en ese momento, permitiendo de esta forma de comprobación periódica de los componentes del sistema. Al igual que en el caso anterior, es posible la desactivación de esta función.

3. `bsmEventForwarderDiscriminator`: Permite la activación y desactivación del filtrado de *traps* y registro de notificaciones, haciendo que el BSM actúe como un EFD (*Event Forwarding Discriminator*, Discriminador de Envío de Eventos) al estilo de la gestión OSI.

4. `bsmLogFile`: Permite cambiar el fichero donde se realiza el almacenaje persistente de los eventos registrados, lo que puede ser útil para el operador.

Así mismo, el BSM mantiene dos tablas que son de gran utilidad a la hora de gestionar la aplicación. Estas son:

5. `bsmConfTable`: Esta tabla contiene el nombre, estado y máquina de los componentes que se han lanzado hasta el momento. Esta tabla se actualiza basándose en las notificaciones que se recibe del resto de componentes y a los accesos que realiza el BSM al demonio de Orbix preguntando por ellos, lo que permite tener que enviar al operador únicamente y de manera asíncrona los cambios que se producen en dicha tabla.

6. `bsmNotificationTable`: Esta tabla contiene el conjunto de las notificaciones recibidas, incluyendo el nombre, el tipo, la hora a la que se recibió y los argumentos de las mismas. Esto permite a un Operador acceder a las notificaciones desde una máquina remota sin tener que obtener el fichero de registro (que se guarda en la máquina donde corre el BSM).

Este conjunto de variables y tablas de gestión no se diferencian en su manejo de las del resto de los componentes. Es decir: El BSM también debe incorporar una interfaz propia de gestión, pero esto no supone mayor problema, dado que el BSM es un componente más de la aplicación de intermediación.

2.3.5. Pasarela SNMP-CORBA

Este es el componente central del conjunto de la arquitectura del sistema de gestión de ABS. Este componente traduce las PDUs (*Protocol Data Units*, Unidades de Datos del Protocolo) de SNMP a peticiones CORBA que son tratadas por los componentes de la aplicación CORBA.

En la primera versión del sistema de gestión [ABS97] (desarrollada para el prototipo de la primera versión de la arquitectura del servicio de intermediación de ABS) se empleó una pasarela muy simplificada. Imponía una relación directa entre los nombres de los grupos SMI y los nombres de los servidores CORBA (de una forma similar a lo que se describe en [Schade96]), no podía manejar componentes individuales de tablas SNMP, y su diseño implicaba la provisión de una interfaz de gestión genérica en todos los componentes de la aplicación CORBA (lo que suponía una gran sobrecarga asociada).

Para la segunda versión del sistema de gestión se adoptaron varios conceptos, principios y procedimientos de JIDM [Open97][OMG98c]. Sin embargo, es importante subrayar que las funcionalidades de JIDM y COSS (*Common Object Services*, Servicios Comunes de Objetos), definidos en [OMG98c], no se han implementado como servicios CORBA separados (servicios que pueden usarse eventualmente por distintas pasarelas que soporten escenarios de SNMP a CORBA o viceversa). En vez de esto, las funcionalidades

necesitadas se han implementado en los diferentes componentes de la pasarela. Esto implica que otros potenciales clientes no pueden usar estos servicios, pero, al mismo tiempo, simplifica el proceso de desarrollo y mejora la eficiencia (reduciendo las interacciones CORBA necesarias en escenarios normales de gestión). Todos estos aspectos se explicarán en detalle durante las siguientes subsecciones.

2.3.5.1. Pila SNMP y el Agente Extensible.

El agente extensible es el componente que recibe todas las peticiones SNMP del sistema de gestión. Su función es, mediante alguna información de configuración, redirigir dichas peticiones a otros componentes *software* específicos capaces de procesarlas. El objetivo es la separación de peticiones dirigidas a la red, aplicaciones del sistema y gestión del servicio. Dentro de la arquitectura global de ABS, el agente extensible hace las funciones del MSA, dado que permite las comunicaciones entre el BSM y la OT.

Para este componente, se ha empleado un agente extensible SNMP escrito en Java, lo que supone un gran punto a su favor para facilitar su integración con el resto del sistema de intermediación. En concreto se ha escogido el implementado por Advent Network Management, Inc. [Advent98], empresa líder en herramientas Java de gestión SNMP. Este agente, uno de los primeros de estas características, une al hecho de estar implementado en Java su facilidad de configuración, dado que permite, mediante pequeñas líneas adicionales en la MIB que se le proporciona, saber qué hacer con las peticiones recibidas.

Según la configuración realizada, las peticiones se pueden redirigir a otros agentes SNMP ejecutándose en puertos distintos, empleándose esta aproximación para la gestión de red y sistema, o a otros objetos Java, empleándose esta última para operaciones relacionadas con la gestión de la aplicación de intermediación de ABS como se describirá más adelante. También permite otras opciones como pueden ser ejecutar algún comando del sistema o leer los datos de algún fichero estático.

Dichas líneas de configuración se incluyen como comentarios SMI pero con un formato especial. Así, la misma MIB poseía dos funciones: La normal, de definición de información de gestión, y la secundaria pero no menos importante, para discriminar peticiones que se deben tratar de distinta manera según su destino.

Un ejemplo de este sistema de configuración se puede ver en el siguiente cuadro, en donde se muestra algunos fragmentos del fichero que se le pasaba al agente extensible.

```
      mib-2       OBJECT IDENTIFIER
--  AGENTCLAUSE
```

```
--       "
--            PROXY-COMMAND:
--            AGENT-HOST: localhost
--            AGENT-PORT: 30000"
--     END AGENTCLAUSE
       ::= { mgmt 1 }
       bsmTimeBetweenDaemonCalls     OBJECT-TYPE
       SYNTAX      INTEGER
       ACCESS      read-write
       STATUS      mandatory
       DESCRIPTION
                   "Time between calling Orbix daemon to obtain the
            activeServers"
-- AGENTCLAUSE
--     "
--            CLASS-COMMAND:
--            CLASS-NAME:ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--            INSTANTIATE: at-start"
-- END AGENTCLAUSE
       ::=   {bsm 2}
```

Cuadro 12. ***Ejemplos de configuraciones***

El primero de ellos se refiere a que hay que redirigir a un agente situado en el puerto `30000` todas las peticiones que se realicen cuyo OID (*Object Identifier*, Identificador de Objeto) comience por el de la MIB-II. El segundo, que hay que redirigir todas las peticiones que recaigan sobre la variable `bsmTimeBetweenDaemonCalls` a la clase Java `ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher`, que se encarga de repartir las peticiones entre los componentes de la aplicación.

Otro tema que debe ser tenido en cuenta en este punto es la pila SNMP. Esta pila, escrita también en Java e incluida en el paquete del Agente Extensible de Advent, se encuentra en la pasarela SNMP-CORBA, y se utiliza en las comunicaciones entre Gestor y Agente.

Las comunicaciones que se realizan en la dirección que va del Gestor al Agente, la pila SNMP, escrita también en Java, queda oculta para el desarrollador, pues el Agente Extensible quien la utiliza. Sin embargo, en la dirección contraria, del Agente al Gestor (lo que sería una *trap* SNMP), sí que se hace uso directamente de la pila SNMP por parte del desarrollador para implementar la funcionalidad de traducción de *traps*. En cualquier caso, dicha pila de

comunicaciones es absolutamente necesaria para el funcionamiento de la pasarela SNMP y del componente MSA.

2.3.5.2. Repartidor de operaciones de gestión

Este componente recibe las peticiones de operaciones de gestión del agente extensible, localiza el componente del sistema de intermediación que puede satisfacer la petición, redirige la petición como operaciones CORBA, recoge los resultados y los devuelve al agente extensible de forma que se pueda generar una respuesta SNMP.

El Repartidor de operaciones de gestión usa el servicio de nombres de CORBA para localizar los objetos CORBA que mantienen la información relacionada con la operación SNMP recibida. Los objetos CORBA que mantienen la información se registran en el servicio de nombres de acuerdo a las reglas contenidas y explicadas en [OMG98c] y estudiadas en el capítulo que se ha dedicado a la Gestión de Aplicaciones Distribuidas, que están fuertemente relacionadas con la forma en que se traducen las MIBs SNMP a interfaces IDL mediante los algoritmos de JIDM-ST (los grupos SNMP y las entradas de tablas SNMP se traducen a interfaces IDL independientes). Sin embargo, un servicio de nombres SNMP separado como el que se define en [OMG98c] no se ha desarrollado y la funcionalidad añadida que este servicio debiera dar (por ejemplo: ordenar las referencias de objetos como filas de tablas SNMP) se realiza dentro del Repartidor de operaciones de gestión. Este hecho implica una pérdida de flexibilidad (las capacidades del servicio de nombres SNMP no se pueden usar por otros componentes) pero simplifica el proceso de desarrollo.

El servicio de repositorio SMI tampoco se ha implementado. En vez de acceder a este servicio par obtener los OID SNMP e información SMI necesaria, la información almacenada en el fichero de la MIB SMI se accede directamente a través de la API Java de SNMP.

Las siguientes figuras muestran de forma esquemática como son los sistemas de obtención de información sugeridos por JIDM-IT y los empleados realmente.

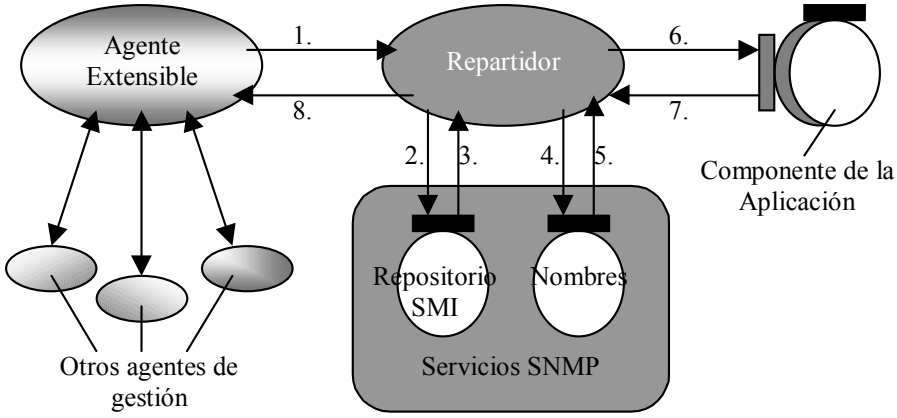


Figura 43. ***Obtención de información con los servicios propuestos por JIDM-IT***

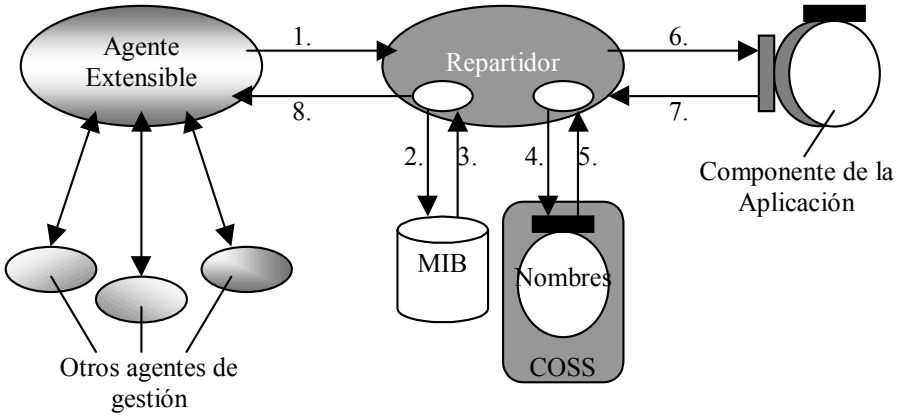


Figura 44. ***Obtención de información con el sistema adoptado finalmente***

Ambas figuras son bastante similares, diferenciándose únicamente en la forma de obtener la información para acceder al componente de la aplicación. La secuencia que se sigue es la siguiente:

1. El agente extensible envía una petición (ya sea un *get*, *getnext* o *set*) al objeto repartidor.

2. Éste solicita el nombre de la variable cuyo OID ha recibido. En el primer caso se realiza mirando en el servicio de repositorio de MIBs propuesto por JIDM. En el caso que atañe, mirando directamente en la MIB cargada previamente. El paquete de Advent proporcionaba un analizador de MIB que realizaba este trabajo.

3. Se devuelve el nombre de la variable.

4. El Repartidor de Operaciones solicita la referencia al objeto que soporta la variable cuyo nombre e índice ha obtenido, basándose en una jerarquía de nombres preestablecida y expuesta en la Traducción de Interacciones de JIDM. En el primer caso, se utiliza un servicio de nombres específico de SNMP que ordena los registros que posee en orden lexicográfico. En el segundo caso se usa el servicio de nombres común de CORBA, y se ha implementado una función en el Repartidor que realiza la ordenación.

5. Se devuelve la referencia al objeto que mantiene la variable.

6. Se envía una petición al objeto que mantiene la variable una vez se ha obtenido su referencia. Para ello, se accede a su interfaz de gestión, o en el caso de ser una variable de una tabla, a la interfaz de gestión propia de dicha tabla.

7. Se obtiene el valor de la variable solicitado, si es una operación de *get*, o nada, en una operación de *set*. En caso de algún error se enviaría una excepción.

8. Se devuelve el resultado al Agente Extensible

Además, la función de reparto se decidió subdividir para dos casos, cada uno de los cuales está contenido en una clase Java diferente a invocar por el Agente extensible:

1. El primero se refiere a operaciones con variables simples.

2. Y un segundo caso, referido a operaciones relacionadas con las variables contenidas en tablas.

Esta división fue debida al hecho de que el tratamiento que se da a una operación *getnext* es distinto para variables simples y variables tabuladas, puesto que en este último caso hay que tener en cuenta la forma de recorrer una tabla.

2.3.5.3. Interfaz de Notificaciones

El punto de acceso del resto de componentes al sistema de gestión para comunicarse con el gestor de manera asíncrona es la Interfaz de Notificaciones. En ella se engloba una serie de funciones que se ejecutan al llamar a dicha Interfaz.

La primera de ellas es la Gestión de Notificaciones, que es en sí misma la que maneja la interfaz que recibe notificaciones de otros objetos. Una vez recibe las notificaciones se encarga de redirigirlas al Traductor de *Traps* y al Gestor del Registro haciendo sendos filtrados previos basándose en la información que posee sobre el estado del resto de los

componentes. El sistema de gestión de ABS no usa ni el servicio de notificaciones SNMP descrito en [OMG98c] ni el servicio genérico de eventos de COSS [OMG97e]. El gestor de notificaciones del sistema de gestión de ABS implementa una aproximación de empuje (*push*) pero sin usar distintos canales de eventos para distintos tipos de eventos. En vez de esto, el único servicio de notificaciones recibe todas las notificaciones de todas los componentes de la aplicación y los reenvía a un único gestor del sistema (mediante el Traductor de *Traps*) y los registra en el fichero que se decida.

El gestor de notificaciones usa notificaciones con tipos de datos concretos, definidas de una forma similar a como se ha propuesto en el actual trabajo de OMG de un servicio de notificaciones CORBA [OMG98a]. La estructura de datos ya se ha definido en el capítulo anterior, por lo que no se vuelve a hacer hincapié en ella. La aproximación utilizada en un primer momento no tenía en cuenta este servicio y en vez de esta estructura de datos usaba únicamente una cadena de caracteres en la que se separaba por "`:`" los distintos argumentos a enviar. Sin embargo, esta falta de definición podía provocar problemas de comprensión, por lo que, sumado al intento de estandarizar en lo posible todos los servicios de gestión, se modificó por la que está actualmente en uso. La interfaz IDL finalmente implementada por el gestor de notificaciones es la que muestra el cuadro siguiente.

```
Interface NotificationManager {
      void notification(in Notification notification_info)
            raises (NotificationException);
};
```

Cuadro 13. ***Interfaz de notificaciones***

El filtrado que se realiza es distinto para el Traductor de *Traps* y para el Gestor del Registro, pero en ambos casos depende únicamente de que el filtrado esté habilitado mediante la variable de gestión `bsmEventForwarderDiscriminator`. En cualquier caso, el filtrado no es configurable, sino que está limitado a los algoritmos que se describen en el siguiente párrafo.

En el caso del discriminador de *traps*, no se realiza traducción y envío de *traps* cuando se cumpla:

1. Que está habilitado el filtrado de eventos.
2. Que el tipo de evento no está definido en la MIB, o bien, si está definido ocurre que es un evento redundante:
   2.1. O bien es un evento indicando el arranque

2.2. O la caída de un objeto de la aplicación de Intermediación del que ya se tenía conocimiento, al estar en la tabla `bsmConfTable`.

En cualquier caso, los eventos generados por el propio Gestor de Notificaciones siempre se reenvían, puesto que esto posibilita una monitorización asíncrona que se comentará más tarde.

En el caso del discriminador de registros ocurre un caso similar, pero con ligeras diferencias: Se admite cualquier tipo de evento, sea conocido o no, y el Gestor de Notificaciones no tiene ningún tratamiento especial. El filtrado también implica que la tabla de configuración se pueda modificar ante el arranque o caída de un componente para mantenerla actualizada.

Una vez se ha realizado el algoritmo de filtrado, El Traductor de *Traps* es el componente de la pasarela responsable de traducir las notificaciones a *traps* SNMP usando los servicios de la pila SNMP, como ya se adelantaba anteriormente. La traducción se realiza de la siguiente manera: Se desencapsula la estructura de la notificación, extrayendo el valor de las variables de tipo `Any`, tantas como número de argumentos posea dicha notificación.

Así mismo, tras el filtrado de la información a registrar, se realiza la conversión de la notificación a texto plano que se almacenará por dos medios distintos: Por una parte, se almacena de forma persistente en el fichero que indique la variable `bsmLogfile`. Por otra parte, se almacena en memoria para cuando el operador desee acceder a la tabla `bsmNotificationTable`.

La figura siguiente muestra, mediante un diagrama de flujo de datos, la secuencia de operaciones descrita.

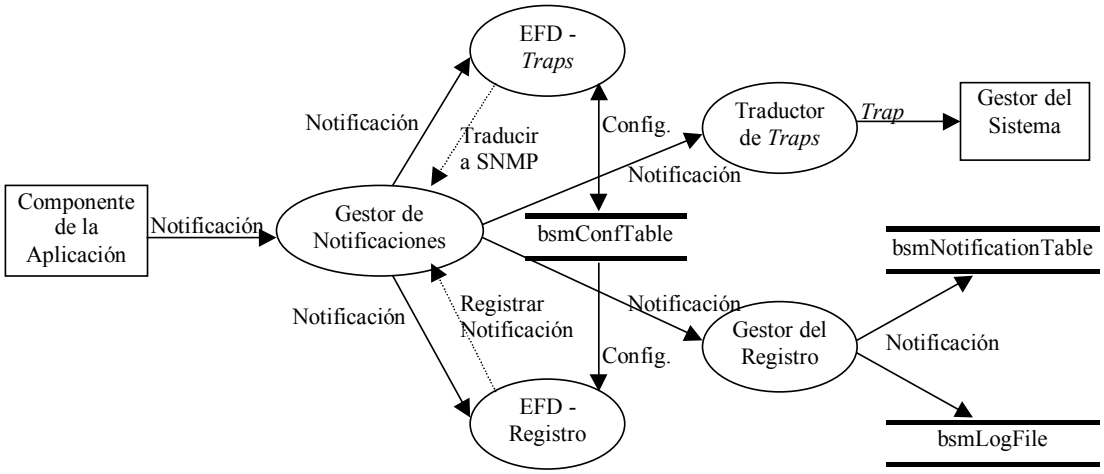


Figura 45. ***Diagrama de flujo***

2.3.5.4. Comprobación del Estado de los Componentes

A la hora de diseñar el sistema se decidió agrupar las funciones que integraban la Interfaz de Notificaciones (Diseminación de Notificaciones, Control de Notificaciones, Registro y control del mismo) con la función de Comprobación del Estado de los componentes del Intermediario en un único componente CORBA de la aplicación de Intermediación: Aquel que se registra como BSM y mantiene la información descrita en el punto 2.3.4.1, aunque realmente sólo sea una parte del sistema de gestión.

Desde un punto de vista de Ingeniería del *Software*, esto supone que la independencia funcional de este objeto se reduzca, pero la sobrecarga de comunicaciones que podía imponer el uso de varios objetos CORBA en el sistema de gestión no era razonable. Fue por ello por lo que se prefirió agrupar todas estas funciones en un único objeto. Por otro lado, el acoplamiento de datos no aumenta tanto ni tampoco se reduce en gran medida la cohesión.

La función de Comprobación del Estado de los Componentes chequea periódicamente, según se le indique con la variable `bsmTimeBetweenDaemonCalls` qué componentes de la aplicación ABS están lanzados y corriendo, manteniendo la tabla `bsmConfTable` basándose en esta información. Esta tabla es accesible a través de la interfaz de gestión del BSM. Además, el monitor del estado de los componentes de la aplicación produce notificaciones para informar al gestor del sistema sobre los cambios de estado de dichos componentes.

El algoritmo es similar al que se emplea en el filtrado de notificaciones y es el que sigue:

1. Se llama al demonio de Orbix, descrito en el segundo capítulo, solicitándole información sobre los servidores que están actualmente corriendo sobre CORBA.

2. La información obtenida se compara con la tabla actual.

   2.1. Se busca si hay nuevos servidores, en cuyo caso se trata de acceder a su interfaz de gestión para determinar si es un servidor gestionable o no, y la máquina donde se encuentra.

   2.2. Se busca si hay servidores desaparecidos, borrando la información que hubiera sobre ellos.

3. Se actualiza la tabla.

2.3.6. Interfaces CORBA de gestión

Esta sección trata de describir el método empleado para implementar las interfaces de gestión de cada uno de los componentes, a las que accede el repartidor de operaciones de gestión para obtener o cambiar el valor de determinadas variables de gestión mantenidas por un determinado componente de la aplicación gestionada. En el sistema de gestión se adopta la aproximación de la arquitectura de gestión TINA [TINA94] que también se sugiere en [OMG98c] para la instrumentación de la MIB SNMP en la aplicación CORBA. En esta aproximación, los objetos computacionales ofrecen dos tipos de interfaces: Una interfaz funcional y otra de gestión. El capítulo de Gestión de Aplicaciones Distribuidas hace un estudio teórico sobre este procedimiento.

Como el lenguaje IDL de OMG no soporta objetos con múltiples interfaces, se ha establecido una relación de herencia en Java (el lenguaje de implementación) entre las clases que implementan las interfaces funcionales y de gestión obteniendo una clase con los métodos de ambos tipos de interfaces. Por tanto, las interfaces de gestión están implementadas por las clases Java llamadas *management handlers* (*manipuladores* de gestión) y de las que un desarrollador de un componente de una aplicación sólo debe derivar su correspondiente clase Java implementada del manipulador adecuado. Además, este hecho ayuda a ocultar el manejo del acceso de información de gestión del desarrollador de la aplicación.

El *manipulador* de gestión contiene atributos que se corresponden con las variables de gestión de su grupo de la SMI. Estos atributos, cuyos valores se acceden mediante operaciones de la interfaz de gestión implementada por el manipulador se pueden modificar también por el código de la clase implementada de un componente particular de la aplicación. Así, los desarrolladores de la aplicación toman el control de la información de gestión sin ser conscientes de cómo se accede a ella.

En una fase inicial del proyecto no se disponía de la documentación que presenta los estándares de JIDM sobre Traducción de Interacciones[OMG98b], [OMG98c]. Por ello, se decidió establecer para todos los componentes de la aplicación una interfaz común que permitiera realizar operaciones sencillas: Obtener y actualizar el valor de variables mediante el uso de dos funciones, que se muestran en el siguiente cuadro. Éstas guardan cierto parecido con aquellas llamadas `get_property_value` y `define_property` de la interfaz IDL `PropertySet` definida dentro del ámbito del servicio CosProperty de COSS [OMG97f].

```
Interface Management {
      any get(in string oid)
```

```
            raises (ManagementException);
      any set(in string oid, in any value)
            raises (ManagementException);
};
```

Cuadro 14. ***Interfaces de gestión iniciales***

El uso de esta interfaz daba sencillez, pero por contra suponía que, para obtener el valor de una celda de una tabla, había que obtener la totalidad de la tabla, debido a que las interfaces eran tan genéricas que los identificadores de objeto (OIDs, *Object Identifiers*) únicamente pasaban el nombre de la variable a tratar, y no su posición dentro de una tabla.

Finalmente, tras disponer de los estándares de JIDM que tratan la parte SNMP de la Traducción de Interacciones [OMG98c], elaborados por el equipo de Subrata Mazumdar, de los laboratorios Bell en la compañía Lucent Technologies, se decidió dar un nuevo rumbo a las interfaces de gestión.

Las interfaces de gestión para los distintos componentes de la aplicación se obtienen aplicando a la MIB SNMP definida para ABS los algoritmos de la Traducción de Especificaciones de JIDM. Para este paso se ha usado el traductor de SMI a IDL desarrollado por el equipo antes mencionado [Mazumdar97].

Así, cada *manipulador* de gestión se deriva a su vez de la clase Java `clientBSM` que implementa métodos complementarios para enviar notificaciones al gestor de notificaciones. Estos métodos se encargan de la encapsulación de notificaciones y su envío al gestor de las mismas, y han sido implementados únicamente para cumplir el principio de que el desarrollo de los diferentes componentes de la aplicación no debía verse afectado por la funcionalidad de gestión. Así, únicamente invocan a un método que se encarga de todo. Se definieron varios métodos para cubrir varios casos:

1. `void sendNotification (String eventType)`. Este método se puede usar para eventos que no tengan parámetros.

2. `void sendNotification (String eventType, EventProperty [] arguments)`. Este otro se debería usar en notificaciones con parámetros.

3. `void sendNotification (Notification notification)`. El último método usa toda la estructura de datos necesaria para enviar una notificación al BSM, preocupándose únicamente de realizar las comunicaciones pertinentes.

En el último caso, el `eventName` del `EventHeader` se puede modificar en la estructura. En los otros, cada método hace uso de la variable `nameSrv` que ha sido inicializada al nombre del componente que lo utiliza. Por ejemplo, para el QEE `nameSrv="QPGSrv"`. En cualquier caso, esta variable se ha definido pública por si es necesario modificarse.

A su vez, esta clase, `clientBSM` se deriva de la clase Java `SmiEntryImplementation` la cual implementa la interfaz IDL `SNMPMgmt::SmiEntry` definida en [OMG98c]. Esta interfaz permite el acceso a la información de gestión a través de métodos genéricos (y por tanto, eludiendo el uso de la interfaz de invocación dinámica de CORBA en la pasarela SNMP-CORBA si fuera necesario). Esta interfaz hereda la `CosProperty::PropertySet`, ya nombrada anteriormente, con lo que el método para obtener y actualizar variables no cambia significativamente.

La clase `SmiEntryImplementation` es también responsable de registrar el objeto CORBA correspondiente en el servicio de nombre de CORBA, de forma que la pasarela SNMP-CORBA pueda localizar el objeto CORBA que soporta una variable SMI en particular.

La figura 46 muestra un diagrama de clases UML (*Unified Modelling Language*, Lenguaje de Modelado Unificado) describiendo la relación entre las clases Java implementadas y sus correspondientes interfaces IDL para la instrumentación de la información de gestión mantenida por el componente QPG de la aplicación de intermediación de ABS.

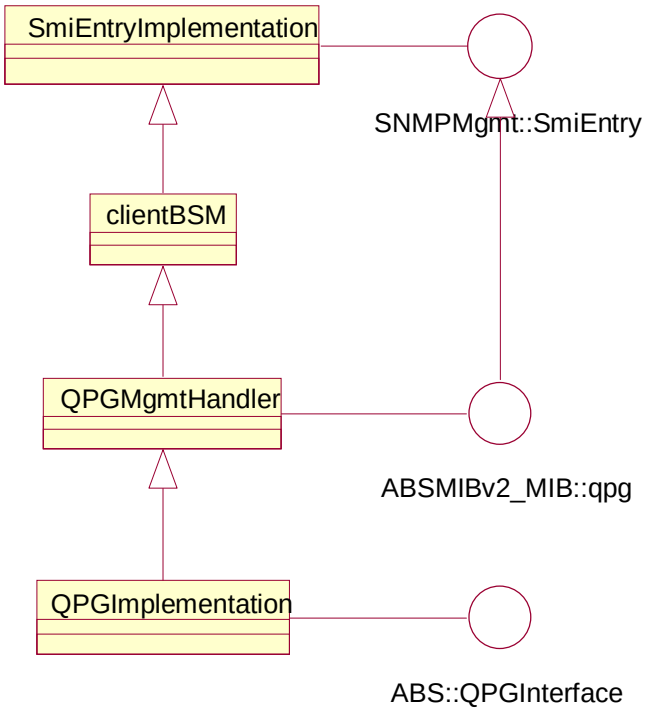


Figura 46. ***Diagrama de clases de gestión del QPG***

Sin embargo, el gran cambio que supone usar la Traducción de Interacciones de JIDM está en el manejo de tablas. Con JIDM cada fila de una tabla es un objeto separado del resto, con lo que es posible acceder a una celda en concreto de una fila sin tener por ello que acceder a la totalidad de la tabla.

Para que los desarrolladores de las implementaciones de los distintos objetos tuvieran acceso a las tablas se planteó una estructura en la que el manipulador poseyera entre sus atributos a las tablas, que a su vez eran secuencias de filas, las cuales heredaban la clase `SmiEntryImplementation`. Un ejemplo de ello es el diagrama de clases Java del BSM. Nótese que las tablas no heredan de `clientBSM`, al no ser en sí mismas emisoras de notificaciones. El `BSMMgmtHandler` tampoco hereda de clientBSM, al ser el BSM el componente que posee precisamente al Gestor de Notificaciones.

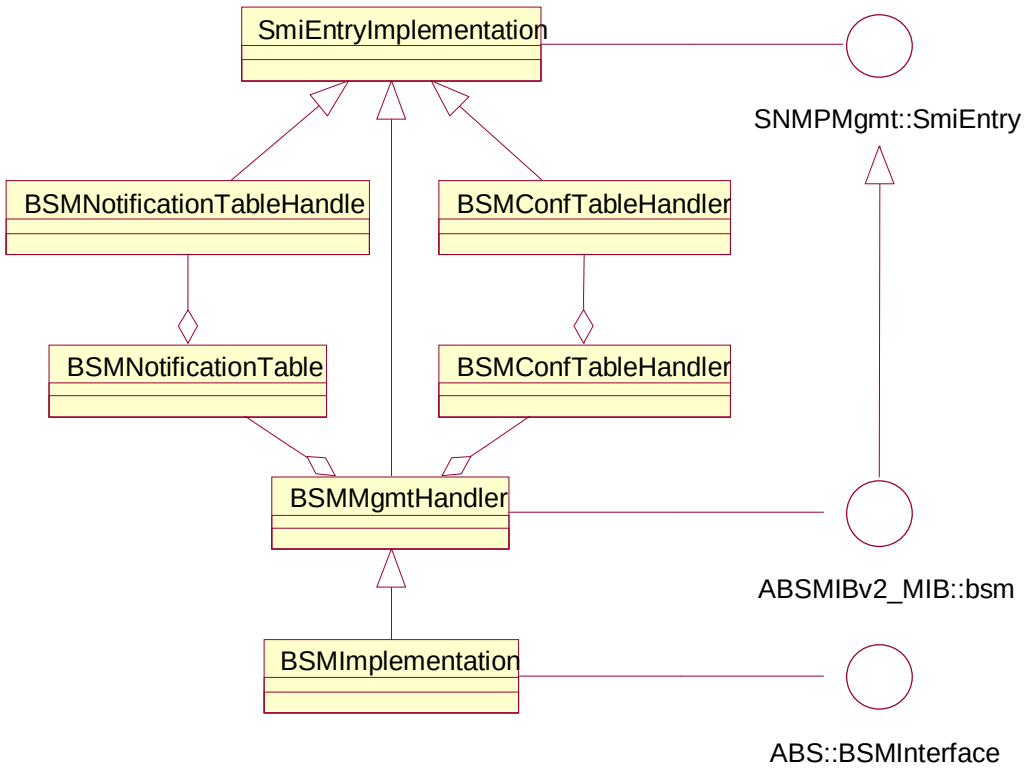


Figura 47. ***Diagramas de clases de gestión del BSM***

Otra funcionalidad que cumplen los manipuladores de gestión es que cuando se ejemplarizan las clases Java de dichos manipuladores, éstas generan una notificación que

indica su activación al Gestor de Notificaciones, el cual decide qué hacer con ellas, basándose en las configuraciones posibles estudiadas con anterioridad. Estas notificaciones, al igual que las del BSM, son configurables y, aunque al ejemplarizar las clases por defecto envían una notificación, se puede hacer que se envíen periódicamente. Esto podría ser útil en plataformas de CORBA que no incorporen una función que indique los servidores que están corriendo sobre ella, pues así, serían los propios servidores quienes se estuvieran validando continuamente respecto a su estado. La figura 48 muestra un diagrama de interacción referente a este último párrafo.

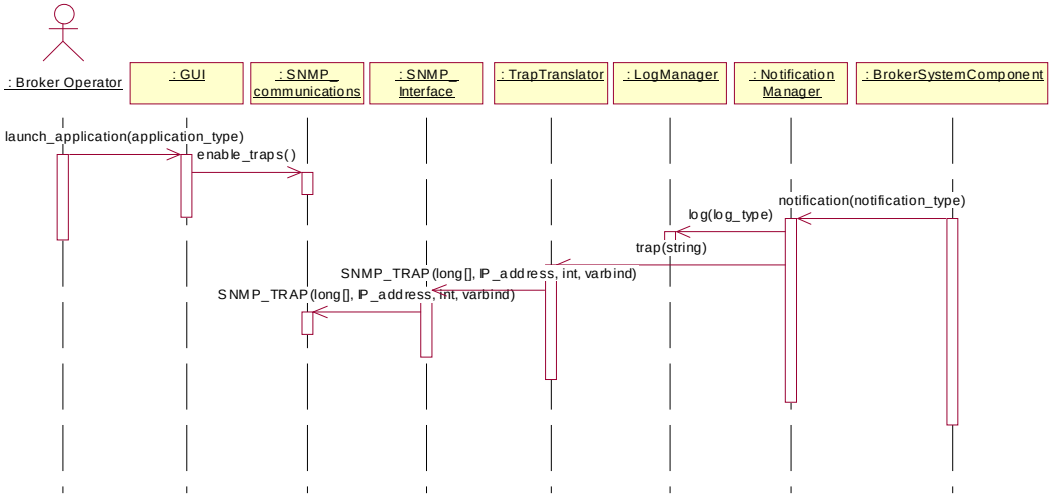


Figura 48. Traps *de arranque*

2.3.7. Consola de Gestión del sistema

2.3.7.1. Introducción

Como ya se ha comentado anteriormente, el sistema de gestión no debía ser implementado desde cero, sino reutilizando la mayor cantidad de componentes existentes de otros sistemas de gestión cuando fuera posible. Además, la gestión de la aplicación se tenía que integrar con aspectos de la gestión de la red y el sistema. Por todo ello, para el sistema de gestión de ABS, se desarrolló la Consola de Gestión SNMP del Operador del sistema usando la plataforma de gestión *Scotty* [Schönwalder95]. Esta se estudia en el siguiente subapartado.

Sobre esta plataforma, se ha desarrollado un conjunto de pequeñas aplicaciones de gestión para adaptar su funcionalidad a la gestión de la aplicación del sistema de intermediación de ABS.

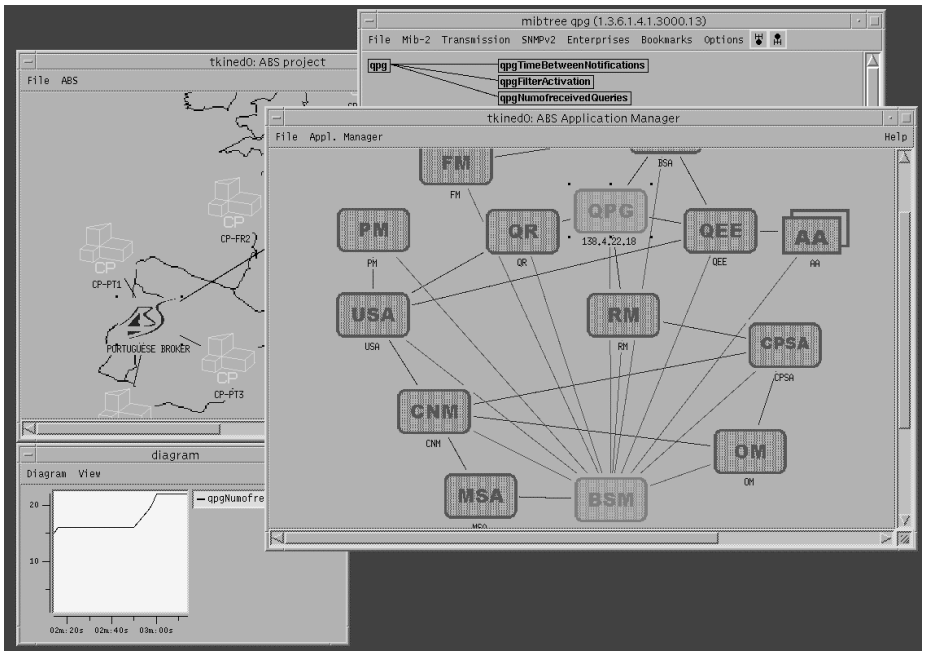


Figura 49. ***Consola de gestión del sistema ABS***

Desde el punto de vista del operador, el gestor del sistema de ABS desarrollado puede realizar las siguientes tareas:

1. Visualizar la MIB de ABS. Cada petición SNMP se reenviará por la pasarela SNMP-CORBA hacia el componente apropiado del sistema de intermediación de ABS.

2. Monitorizar gráficamente variables de la red, sistema y aplicaciones.

3. Monitorizar el estado de los componentes de la aplicación.

En los siguientes puntos se irá desarrollando el diseño de la Consola de Gestión.

2.3.7.2. *Scotty*

Scotty es el nombre de un paquete *software* que permite implementar *software* de gestión de red específico para un lugar concreto usando APIs (*Application Programming Interfaces*, Interfaces de Programación de Aplicaciones) de alto nivel y basadas en cadenas de caracteres. El *software* se basa en TCL (*Tool Command Language*, Lenguaje de Comandos de Herramientas) [Welch95], lo que simplifica el desarrollo de *scripts* (código de alto nivel interpretable que no dependen de la arquitectura de la máquina donde corren) de gestión de red. La distribución del *Scotty* incluye dos componentes principales. El primero es la Extensión de TCL Tnm, que proporciona acceso a las fuentes de información de gestión de red. El segundo componente es el editor de red *Tkined*, que proporciona un marco para un

sistema extensible de gestión de red, y que ha sido usado para la presentación gráfica de la Consola.

La Extensión de TCL Tmn permite acceder a las fuentes de información de gestión de red desde TCL. La Extensión Tmn soporta los siguientes protocolos:

1. SNMP (SNMPv1, SNMPv2c, SNMPv2u incluyendo el acceso a las definiciones de MIB)
2. ICMP (`echo`, `mask`, `timestamp` y peticiones `traceroute` udp/icmp)
3. DNS (ver registros `a`, `ptr`, `hinfo`, `mx` y `soa`)
4. HTTP (lados del servidor y del cliente)
5. SUN RPC (servicios portmapper, mount, rstat, etherstat, pcnfs)
6. NTP (peticiones versión 3 modo 6)
7. UDP (enviar y recibir datagramas UDP - sin funcionalidad de canales)

Se incluye comandos adicionales para simplificar la implementación de aplicaciones de gestión de red:

1. El comando `netdb` permite acceder a las bases de datos de la red local (nombres de máquinas y direcciones IP, servicio de nombres, nombres de red, nombres de protocolo, servicio de nombres *Sun* RPC).
2. El comando `syslog` permite enviar mensajes a la herramienta local de registro.
3. El comando `job` simplifica la implementación de procedimientos de monitorización o controlo que necesitan ejecutarse a intervalos regulares.

Se incluye un visualizador gráfico de MIBs SNMP que permite explorar datos de una MIB. Este examinador es una útil herramienta de gestión así como para desarrollar aplicaciones de gestión.

La Extensión TCL Tnm es usada por los vendedores de *software* de gestión basado en SNMP para implementar conjuntos de tests de regresión para sus productos. Estos tests son fáciles de implementar, puesto que la extensión Tnm puede interpretar un papel dual, permitiendo crear prototipos de agentes SNMP escritos en *scripts* TCL.

El editor de red *Tkined* proporciona un marco para una plataforma de gestión de red extensible. Las Extensiones se escriben normalmente en TCL basado en la extensión Tnm.

Esto permite integrar herramientas de gestión específicas del lugar o la red a costes bajos. La API Tcl permite controlar cualquier asepcto del editor lo que hace posible incluso automatizar interacciones de usuario con *scripts* TCL.

Se puede mantener mapas de red usando las características de edición que proporciona el Tkined o usando herramientas automatizadas. La distribución contiene aplicaciones para descubrir redes IP, soportar el proceso de diseño de red, resolver problemas de redes IP usando SNMP junto con otras herramientas estándar (como `traceroute`) y monitorizar el estado de la red usando SNMP o las RPCs de SUN seleccionadas.

El paquete *Scotty* está disponible de manera gratuita, y corre en la mayoría de las plataformas UNIX. Para que funcione es necesario tener un intérprete de TCL/TK.

2.3.7.3. Modificaciones a *Scotty/Tkined*

A la hora de adaptar el paquete al caso particular de ABS se llevaron a cabo varias modificaciones, siendo estas de dos tipos:

1. **Mejoras del entorno gráfico**. En este apartado se incluye todas aquellas mejoras que han permitido modificar la GUI.

   1.1. Así, se decidió que la Consola debía arrancar con la aplicación de gestión de ABS, y no como el editor de red que realmente es.

   1.2. Además, todas aquellas funcionalidades que el *Tkined* incluía en sus menús que no eran de utilidad para el Operador se eliminaron.

   1.3. Así mismo, se añadió al código la posibilidad de poder acceder a funciones adicionales al pinchar el botón izquierdo del ratón sobre determinados iconos tales como visualizar la MIB asociada a dichos iconos. Estas funciones son accesibles también desde un menú especial de ABS añadido al entorno, pero su acceso desde el icono lo hace más intuitivo.

2. **Modificaciones para posibilitar nuevas funcionalidades**. Aquí se incluye los cambios en el código para poder acceder a nuevas funcionalidades, añadidas para el proyecto ABS. Así, entre estas modificaciones se incluyen las que posibilitan la monitorización gráfica en otra ventana, la visualización de la MIB específica de ABS. También está la posibilidad de lanzar una nueva ventana de *Tkined*, hecho que se utilizará a la hora de llevar a cabo la gestión de la aplicación.

Además de estas modificaciones en el código del *Scotty*, se añadió una serie de aplicaciones para facilitar la gestión de ABS, que serán comentadas a continuación. Se redactó así mismo un manual de usuario, que se incluye en la sección de planos (y que puede ser también consultado en http://babel.dit.upm.es/~jlopez/pfc/usermanual/).

2.3.7.4. Gestión de la aplicación

En este apartado se quiere dar una visión global en más detalle de la Consola. Esta se decidió dividir principalmente en dos ventanas, con funcionalidades y aplicaciones diferentes:

1. **Ventana Principal**: Esta ventana es el punto de arranque del Operador. En ella se muestra el mapa de Europa con iconos situados en aquellos puntos donde se encuentre un Sistema de Intermediación ABS. Estos iconos son puntos de acceso a cada uno de dichos sistemas, con lo que se podría desde una misma Consola gestionar todos los Sistemas de Intermediación, si bien la carga que esto suponga puede resultar excesiva para un buen rendimiento. Sobre estos iconos se puede realizar las siguientes operaciones:

   1.1. **Visualizar las distintas MIBs** mantenidas por el sistema computacional que soporta el Sistema de Intermediación. Por defecto se visualiza la MIB de ABS. Con esto se consigue la integración con gestión de red y sistema, al poder acceder a toda la información de gestión mantenida por el sistema computacional.

   1.2. **Acceder a la ventana de gestión de la aplicación**, que se explica más adelante. Para realizar esta tarea, se lanza un intérprete, proceso que interpreta TCL, ejecutándose otro *Tkined* al que se le ha pasado los parámetros necesarios para que sepa en qué sistema se encuentra dicha aplicación. Esta función supuso un gran esfuerzo, pues la API del *Tkined* por sí misma no permite que se lancen varias ventanas, unas hijas de otras, con lo que hubo que aplicar una argucia que consistía en la interpretación del propio *Tkined*, que está escrito en TCL.

Además, se puede acceder, mediante el menú específico de ABS, a una pequeña ventana de ayuda que explica las posibilidades de dicho menú.

Tanto el aspecto de ésta como otras ventanas se puede ver en el manual de usuario adjunto en la sección de planos.

2. **Ventana de gestión de la aplicación**: Esta ventana es funcionalmente la más importante y a la que se ha dedicado más esfuerzo de desarrollo. Su función es la gestión de la aplicación de intermediación. En ella se representa, mediante iconos, los distintos

componentes de la arquitectura del intermediario. Dichos iconos pueden mostrar en base a su color su estado. También muestran si se ha recibido una *trap* SNMP de uno de dichos componentes. Desde ella se puede acceder a las siguientes aplicaciones:

2.1. **Monitorización del estado del sistema**: Muestra, basándose en los colores de los componentes, rojo, verde y naranja, si los distintos componentes están caídos, lanzados con posibilidad de gestión o lanzados sin posibilidad de gestión. Se definieron distintas políticas de monitorización, que se discuten en el siguiente apartado. En cualquier caso, una ventana muestra el estado de la monitorización.

2.2. **Monitorización de *Traps* SNMP**: Habilita la recepción de *traps* SNMP y, según los argumentos recibidos, obra en consecuencia: Hace parpadear el componente que originó la notificación, y en caso de ser una *trap* que indique el lanzamiento o caída de un objeto, actualiza los datos que sobre él se tiene. A su vez, se genera una ventana de registro, con lo que el operador puede ver la información de las *traps* que le han llegado.

2.3. **Visualización de la MIB**: En este caso, es posible pinchar o seleccionar un icono para ver el grupo de la MIB que mantiene el componente que representa.

2.4. **Configuración de parámetros SNMP**: Permite la configuración de parámetros SNMP, tales como el tiempo de retardo, el tamaño de la ventana o el número de retransmisiones, lo que puede valer para modificar las variables del protocolo de gestión para que responda mejor ante condiciones de alta carga de la red o el sistema, manteniendo la fiabilidad necesaria.

Además, como en el caso anterior, se puede acceder mediante el menú específico de gestión de la aplicación ABS a una pequeña ventana de ayuda que explica las posibilidades de dicho menú.

Desde cada una de dichas ventanas se puede acceder al **visualizador de MIBs**. Esta es una aplicación incluida en el paquete *Scotty* y que ha sido adaptada para el gestor de ABS. Su funcionamiento consiste en que dado un OID (*Object Identifier*, identificador de objeto) ASN.1, muestra todos los nodos que cuelgan de dicho OID. La aplicación está preparada para mostrar MIBs tales como la MIB-II [McCloghrie91b], MIB de Transmisión [Baker92], MIB de SNMPv2 [Case96] y MIBs de empresas particulares, como puedan ser la de *Sun*

[SunSoft94] o la propia de ABS. Además, también hay disponibles menús desplegables que según el objeto sea rama u hoja permite realizar alguna de las siguientes funciones:

1. **Describir variable de la MIB**: Da información sobre las distintas variables, incluyendo su OID, su estado, accesibilidad y descripción, si la posee.

2. **Mostrar variable**: Realizar una operación *get* SNMP sobre una variable, mostrando su valor en caso de éxito.

3. **Mostrar tabla**: Realiza varias operaciones de *getnext* SNMP sucesivas hasta que se muestra una tabla SNMP en su totalidad, de forma tabulada, en filas y columnas, facilitando su lectura.

4. **Avanzar por un nodo**: Realiza varias operaciones de *getnext* SNMP para obtener los valores de todas las variables que hay a partir de ese nodo.

5. **Escribir variable**: Realiza una operación *set* SNMP a una variable con acceso de escritura, para modificar el valor que posee.

6. **Monitorizar variable**: Hace una monitorización gráfica de dicha variable. Esta funcionalidad se explicará más adelante.

7. **Ir a un nodo**: Avanza o retrocede dentro del árbol, pudiendo avanzar hacia los nodos hijos, o bien, retroceder al nodo padre.

Para acceder a toda esta información en el caso de gestión de la aplicación, la monitorización de estado debe estar en funcionamiento, pues de lo contrario, SNMP Tree no puede encontrar la dirección IP donde está el agente para direccionar las peticiones.

2.3.7.5. Políticas de Monitorización

Se realizó un estudio detallado de la política a llevar a cabo para realizar la monitorización del sistema, de forma que fuera lo más eficiente posible, tras el cual se llegó a unas conclusiones, en las cuales se esbozaban un conjunto de políticas, que son las que siguen:

1. **Monitorización desde el dominio SNMP**. Esta fue la adoptada en un primer momento, pues es similar a monitorizar variables, sólo que en este caso se monitoriza toda una tabla de configuración, que se puede obtener invocando a cada uno de los objetos de gestión

cuyos nombres aparezcan en la MIB. Las interacciones llevadas a cabo con esta política son las siguientes:

1.1. Invocación de obtención de la tabla de configuración. Como cada tabla tiene cuatro columnas, las interacciones SNMP se multiplican por este número, además del número de retransmisiones, que en el caso de *Tkined* es tres por defecto. Es decir, en total se realizan doce operaciones *get* SNMP por cada componente.

1.2. Petición de la tabla en el dominio CORBA. El Repartidor trata de establecer comunicación con el componente solicitado. En el dominio CORBA disminuye el número de interacciones, pues en la tabla que mantiene el Repartidor ya se encuentra el nombre del componente y el número de columna.

1.3. Envío de cada una de las respuestas al Repartidor. En caso de que el componente no exista, se obtiene una excepción, con la que se rellenan los dos campos restantes a que no se conoce la máquina donde corre y que el objeto no está corriendo. En caso contrario, el componente está lanzado y al obtener su referencia también se sabe la máquina donde está.

1.4. Envío de las respuestas al Gestor. Como en la primera interacción, habrá tantas respuestas como peticiones se reciban, por lo que pueden ser un total de 12.

A su vez, las interacciones habrá que multiplicarlas por el número de filas, que es lo mismo que el número de componentes, de la tabla. Estas interacciones se pueden ver en la figura 50.

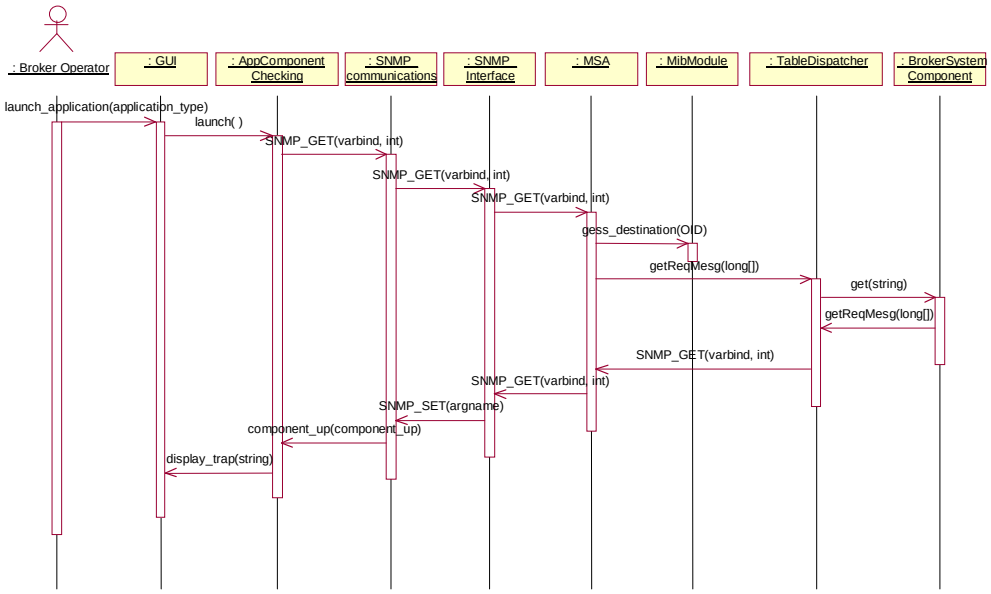


Figura 50. ***Monitorización desde el dominio SNMP***

2. **Monitorización basada en eventos**. Esta segunda se basa en los sistemas de gestión OSI [Stallings93], en que son los agentes quienes envían la información al gestor, y son los EFDs quienes se encargan de discernir entre información importante o redundante. En este caso, cada componente envía al BSM cada cierto tiempo una notificación de que se encuentra lanzado, y éste la reenvía al sistema gestor. A su vez, en el sistema gestor debe existir un proceso que esté pendiente de cuando ha sido la última vez que se ha recibido una notificación, para, en caso de que se tarde mucho, decidir que el componente se ha caído. Todos los tiempos deben ser configurables. En este caso disminuye el número de interacciones. Para cada componente ocurrirá:

   2.1. Establecimiento de comunicación con el BSM, en el dominio CORBA.

   2.2. Envío de la notificación, en el dominio CORBA.

   2.3. Se traduce la notificación al dominio SNMP que la recibe.

   Como esto es para cada componente, estas interacciones habrá que multiplicarlas por el número de componentes lanzados. Las interacciones generadas por un componente se pueden ver en la figura 51.

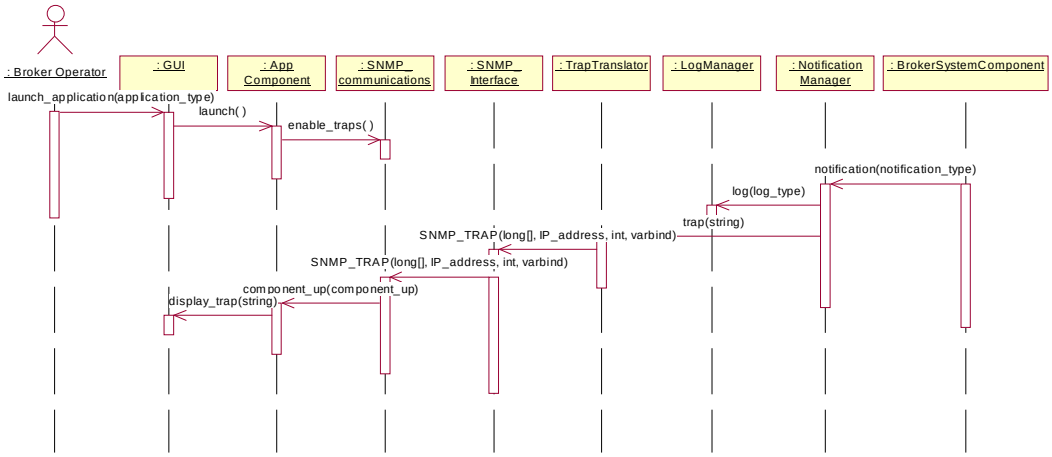


Figura 51. ***Monitorización basada en eventos***

3. **Monitorización desde el dominio CORBA**. Esta tercera establece un objeto que realiza la monitorización desde el dominio CORBA, con intervalo entre monitorizaciones configurable, que accede al servicio del demonio de Orbix y solicita el total de objetos corriendo en esa plataforma. Después mira si esos objetos están en la tabla que mantiene, para, ante un cambio, enviar una notificación al Gestor, para que sepa este cambio. A su vez, el objeto (en este caso, el BSM) debe enviar periódicamente notificaciones al Gestor para que este sepa que sigue vivo y que la información que posee en la Consola no es incongruente con la realidad. Para ello, se debe crear un proceso parecido al anterior, pero que sólo esté pendiente del BSM. Además, al principio deberá solicitar la tabla de componentes activos, pues no la conoce. En este caso, el número de interacciones disminuye. Para todos los componentes ocurrirá:

   3.1. El BSM establece la comunicación con el demonio una sola vez, y se mantiene, y cada cierto tiempo el BSM realiza la petición de la lista de servidores activos.

   3.2. Se devuelve la lista de una sola vez al BSM.

   3.3. En caso de cambios en la tabla que mantiene el BSM, o a intervalos regulares para mostrar su propio estado, se envía una notificación que es traducida a SNMP. Esta última interacción realiza pasos similares a los incluidos en el segundo método de monitorización, basada en eventos, sólo que el Gestor de Notificaciones, Registro y Traducción de Traps son internos al BSM.

Un esquema general de las interacciones se puede ver en la figura 52. Los primeros pasos

mostrados sólo se ejecutan al arrancar, y son debidos a la obtención de la tabla de estados inicial. Los importantes son los que se llevan a cabo en la segunda parte del diagrama.

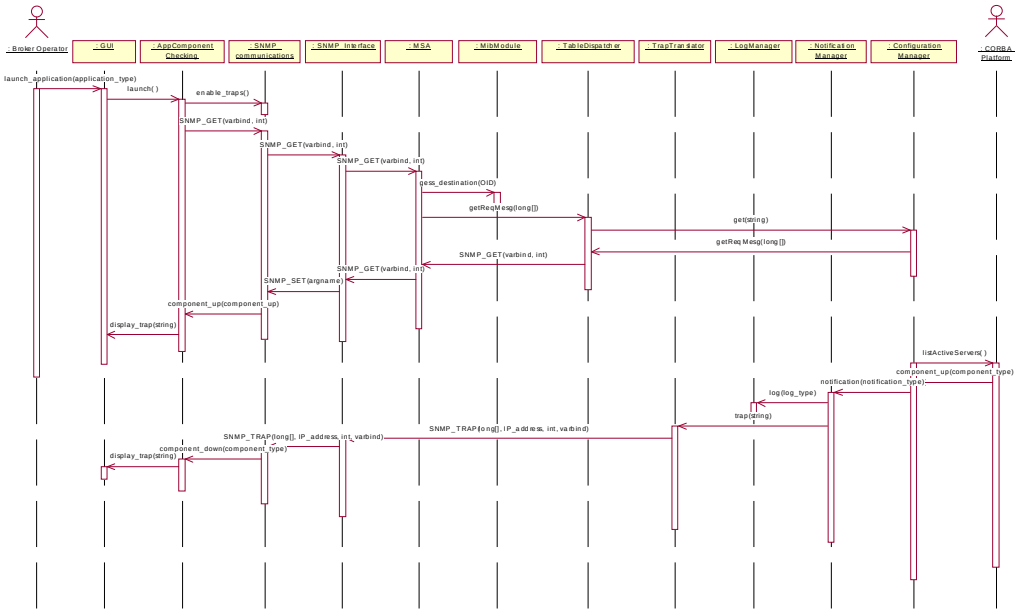


Figura 52. ***Monitorización desde el dominio CORBA***

4. **Monitorización conjunta basada en eventos y desde el dominio CORBA**. Este caso es una particularización del anterior en el que los componentes envían una notificación cuando arrancan, por lo que el BSM no tiene que esperar a realizar la petición de servidores para enterarse de la activación de un nuevo componente. En este caso aumenta el número de interacciones, pero la congruencia de la información de la Consola también es mayor.

5. **Monitorización conjunta desde los dominios SNMP y CORBA**. Como se puede ver, la monitorización que menos interacciones supone es la del dominio CORBA para un estado estable. Sin embargo, esto supone que ante una caída del BSM no se podría tener información del estado del resto de los componentes. Para ello, lo que se ha pensado es que, en este caso, se proceda a cambiar automáticamente a la monitorización desde el dominio SNMP hasta que una *trap* del BSM indicando su relanzamiento conmute de nuevo a la monitorización CORBA.

Cada una de las distintas políticas ha sido implementada, comprobándose que el mejor método de monitorización es el que se hace desde el dominio CORBA, optándose finalmente por la última explicada pues es la que permite una mayor coherencia de información.

2.3.7.6. Monitorización Gráfica de variables de gestión

Una de las opciones que poseía el Visualizador de MIBs era realizar la monitorización de una variable. Esta monitorización es configurable, pudiéndose elegir la ventana tiempo de visión, el intervalo entre peticiones y si se quiere que la monitorización sea diferencial (visualiza deltas ante un cambio) o absoluta (visualiza el valor exacto de la variable). El proceso que supone esto es bastante simple, y consiste en ir haciendo operaciones *get* SNMP de una variable concreta, separando dichas peticiones el intervalo seleccionado. En el caso de la monitorización diferencial se resta el valor anterior del posterior, obteniéndose la diferencia, por lo que las deltas que se obtienen pueden estar multiplicadas por un coeficiente distinto de uno en caso de aumentos súbitos de valor.

Aunque el paquete *Scotty* ya incluía esta funcionalidad se cambió, pues la forma en que se visualizaba era deficiente. Así como no podía haber ventanas hijas, tampoco se podía visualizar gráficas en ventanas hijas, sino que, por el contrario, las gráficas se visualizaban en la misma ventana del *Tkined*. Para realizar esta modificación se usó el paquete gráfico basado en TK BLT [Scriptics98], consiguiéndose que las gráficas de las variables monitorizadas aparecieran en una ventana aparte.

2.3.8. Conclusiones

El empleo del diseño descrito ha posibilitado cumplir los requisitos iniciales. Así, El sistema de gestión hizo uso de herramientas ya existentes en el mercado, tales como el Agente Extensible y la Pila SNMP y la plataforma de gestión *Scotty*. Así mismo, los Agentes de Red y Sistema no se implementaron, sino que se utilizó un agente para máquinas *Sun* [SunSoft94] para la gestión de estos recursos. Todo esto posibilitó un desarrollo más rápido y con menor coste.

Además, se ha conseguido afectar, mediante el uso de los *manipuladores* de gestión, lo menos posible el desarrollo de los diferentes componentes de la aplicación por la funcionalidad de gestión. Esto ha redundado en una mayor productividad, al separar claramente las tareas de cada desarrollador.

Pero el factor protagonista es que, con el presente sistema, el operador de la consola de gestión es capaz de obtener de forma conjunta información de todos los entes que tengan implicaciones en el sistema de intermediación de ABS. Esto se ha conseguido mediante la integración de gestión CORBA con aspectos de la gestión de la red y el sistema.

Un gran ejemplo de ello es poder visualizar la MIB de cada entidad involucrada en la aplicación de intermediación. La funcionalidad de la visualización de la MIB permite acceder a cada una de las variables de gestión, ya sean internas de ABS o no, que mantiene cada sistema computacional. Así, al poder acceder a dichas variables, se puede saber incluso de forma gráfica la carga de los sistemas y de las interfaces de red de dichos sistemas.

En parte, esto se consigue gracias a que cada petición SNMP se reenvía al agente extensible, con lo cual es posible realizar esta integración. Esto también supone que todos los sistemas sobre los que se quiera tener esta integración de gestión deberán poseer un agente extensible que redirija las peticiones al ente adecuado.

Con todo, el sistema diseñado cumple los requisitos presentados en el segundo apartado de este capítulo.

2.4. Implementación del Sistema de Gestión

2.4.1. Introducción

El presente capítulo trata de reflejar el trabajo desarrollado durante la implementación del Sistema de Gestión de la aplicación distribuida de intermediación descrita en el capítulo 1.4.

Dado que únicamente se pretende dar una visión amplia del código, solamente se procederá a describir los ficheros que lo contienen. No obstante, se muestra un ejemplo del mismo en el apéndice B (en concreto el fichero `BSMMgmtHandler.java`).

2.4.2. Descripción de ficheros

Esta sección se ha dividido en dos grupos: El primero contiene los ficheros que pertenecen al dominio del Intermediario y que están todos implementados en Java. El segundo, los ficheros que pertenecen al dominio del Operador, que están escritos en TCL.

2.4.2.1. Dominio del Intermediario

Aquellos ficheros que pertenecen a la implementación del BSM son los siguientes:

1. Clases que implementan el Repartidor de Operaciones de gestión:
   1.1. `BSM.idl`: Define la interfaz IDL `NotificationManager`
   1.2. `AbsDispatcher.java`: Clase que reparte operaciones para variables simples.
   1.3. `AbsDispatcherTable.java`: Clase que reparte operaciones para tablas.
   1.4. `Configuration.java`: Clase para monitorizar los componentes vía SNMP.
   1.5. `NamingContextImplementation.java`: Simula la funcionalidad de ordenación de tablas descrita en [OMG98b] junto con `NcEntryIteratorImplementation.java`.
   1.6. `NcEntryIteratorImplementation.java:` Simula la funcionalidad de ordenación de tablas descrita en [OMG98b] junto con `NamingContextImplementation.java`.
2. Clases que realizan la gestión del sistema: notificaciones, *traps*, registros, monitorizaciones.
   2.1. `BSMserver.java`: Es el servidor CORBA que sustenta al resto de las clases.
   2.2. `BSMImplementation.java`: Es la implementación de la interfaz `NotificationManager`, e incluye además toda la funcionalidad de registro y traducción de traps.

2.3. `AgentProcess.java`: Esta clase implementa un *thread* que realiza las monitorizaciones en el dominio CORBA.

2.4. `NotificationProcess.java`: Esta última clase se encarga de enviar notificaciones periódicas a la consola de gestión para asegurar la existencia del BSM.

Por otro lado se encuentran todos los archivos que implementan las interfaces de gestión:

1. `SmiEntryImplementation.java`: Esta clase implementa la interfaz `SmiEntry`, descrita en el apartado 2.3.6. De ella hereda el `clientBSM.java` y aquellos manipuladores de gestión que no necesiten enviar notificaciones, como puedan ser aquellos que mantienen tablas.
2. `clientBSM.java`: Hereda de `SmiEntryImplementation.java` y facilita la funcionalidad de envío de notificaciones.
3. `AAMgmtHandler.java`: Es el manipulador de gestión del AA, que hereda de `clientBSM.java`.
4. `BAMMgmtHandler.java`: Es el manipulador de gestión del BAM, que también hereda de `clientBSM.java`.
5. `BSAMgmtHandler.java`: Manipulador de gestión del BSA. Hereda de `clientBSM.java`.
6. `BSMConfTable.java`: Mantiene la tabla de configuración de componentes.
7. `BSMConfTableEntryHandler.java`: Esta clase mantiene la información de gestión de una fila de la tabla de configuración. Hereda directamente de `SmiEntryImplementation.java`.
8. `BSMMgmtHandler.java`: Es el manipulador de gestión del BSM, que hereda directamente de SmiEntryImplementation.java, al ser el propio BSM quien posee la funcionalidad de notificaciones. Posee las clases-tabla `BSMConfTable.java` y `BSMNotificationTable.java`.
9. `BSMNotificationTable.java`: Mantiene la tabla de notificaciones recibidas, haciendo la función de registro.
10. `BSMNotificationTableEntryHandler.java`: Esta clase mantiene la información de gestión de una fila de la tabla de notificaciones. Al igual que BSMConfTableEntryHandler.java, hereda directamente de `SmiEntryImplementation.java`.

11. `CNMMgmtHandler.java`: Manipulador de gestión del CNM, que hereda de `clientBSM.java`.
12. `CPSAMgmtHandler.java`: Manipulador de gestión del CPSA, que hereda de `clientBSM.java`.
13. `FMMgmtHandler.java`: Manipulador de gestión del FM, que hereda de `clientBSM.java`.
14. `MSAMgmtHandler.java`: Manipulador de gestión del MSA, que hereda de `clientBSM.java`.
15. `OMMgmtHandler.java`: Manipulador de gestión del OM, que hereda de `clientBSM.java`.
16. `PMMgmtHandler.java`: Manipulador de gestión del PM, que hereda de `clientBSM.java`.
17. `QEEMgmtHandler.java`: Manipulador de gestión del QEE, que hereda de `clientBSM.java`.
18. `QPGMgmtHandler.java`: Manipulador de gestión del QPG, que hereda de `clientBSM.java`.
19. `QRMgmtHandler.java`: Manipulador de gestión del QR, que hereda de `clientBSM.java`.
20. `RMMgmtHandler.java`: Manipulador de gestión del RM, que hereda de `clientBSM.java`.
21. `USAMgmtHandler.java`: Manipulador de gestión del USA, que hereda de `clientBSM.java`.
22. `testBSM.java`: Por último, se incluye una clase que hereda de `clientBSM.java` y que únicamente sirve para probar la interfaz de notificaciones del BSM.

2.4.2.2. Dominio del Operador

En el dominio del operador se encuentran aquellos ficheros que se han utilizado para añadir funcionalidades al *Scotty*, tal y como se contaba en la sección 2.3.7. Estos son:

1. Programas principales:
   1.1. `tkabs`: Este programa arranca el editor de red *Tkined*, pero ejecutando directamente el *software* de gestión de ABS.
   1.2. `service.tcl`: Este *script* contiene el código a interpretar inicialmente.
   1.3. `application.tcl`: Este otro, el que se ejecuta para lanzar la ventana de gestión de la aplicación.
   1.4. `tkison`: Este programa lanza el editor de red *Tkined*, que interpreta el *software* de gestión de la aplicación en una ventana hija.
   1.5. `applmgmt.tcl`: Este último, ejecuta sobre la ventana hija el *software* de gestión de la aplicación.
2. Monitorizaciones:

2.1. `Diagram.tcl`: Este *script* muestra gráficamente las variaciones que experimenta con el tiempo una variable de gestión.

2.2. `graphical_monitor.tcl`: Este código se encarga de realizar el trabajo de monitorización de variables que se visualizarán gráficamente.

2.3. `bsm_monitor.tcl`: Este *script* se encarga de mantener la monitorización desde el dominio CORBA.

2.4. `trap_monitor.tcl`: Este programa se encarga de tratar las *traps* recibidas.

2.5. `ttl.tcl`: Este código se encarga de mantener la monitorización basada en eventos.

2.6. `nice.tcl`: Finalmente, este último realiza monitorización conjunta SNMP y CORBA.

3. Otras herramientas:

3.1. `mibtree.tcl`: Esta herramienta se encarga de la tarea de visualización de MIBs.

4. Programas auxiliares:

4.1. `clearview.tcl`: Pregunta si se debe borrar la pantalla.

4.2. `delete.tcl`: Borra la pantalla.

4.3. `initservicemgmt.tcl`: Realiza algunas tareas rutinarias al arrancar la aplicación de gestión.

5. Mapas

5.1. `servicemap.tki`: Contiene el mapa de europa, con los distintos lugares donde hay un intermediario electrónico.

5.2. `applimal.tki`: Contiene un mapa con los distintos componentes de la aplicación.

2.4.3. Conclusiones

En el presente capítulo se ha descrito superficialmente, enumerando los ficheros que contienen el código desarrollado, la parte de codificación e implementación del presente proyecto.

Los ficheros se han dividido, según el dominio al que pertenecían, en ficheros del Intermediario y ficheros del Operador, escritos, respectivamente, en Java y TCL.

3. Conclusiones y futuras líneas de trabajo

3.1. Introducción

Este capítulo pretende, tras haberse desarrollado en los capítulos anteriores los temas teóricos y metodológicos empleados en la realización del presente proyecto, tratar de dar unas conclusiones al trabajo realizado, así como proponer futuras líneas de trabajo a tener en cuenta para una posible ampliación del presente proyecto.

Para ello, se ha dividido el trabajo en tres partes:

1. **Comentarios sobre el rendimiento**: Esta parte trata de dar una visión resumida del comportamiento del sistema, dando algunos datos sobre su rendimiento que pueden resultar de interés.
2. **Conclusiones**: Este apartado da en sí mismo las inferencias obtenidas al estudiar el sistema implementado.
3. **Futuras líneas de trabajo**: Este último apartado trata de dar futuros caminos de investigación y desarrollo a llevar a cabo una vez se ha concluido el trabajo realizado.

3.2. Comentarios sobre el rendimiento

A continuación se muestra una serie de datos que pueden resultar de interés para entender el comportamiento del sistema. En una primera parte se hará una comparación entre el uso de JIDM y la primera aproximación de interfaces de gestión comentada en el capítulo de Diseño. La segunda parte hará una comparativa entre las distintas políticas de monitorización utilizadas en la consola de gestión.

3.2.1. JIDM

Como ya se comentó en la sección 2.3.6, antes de tener los estándares de Traducción de JIDM [Open97][OMG98b][OMG98c], se utilizó una aproximación muy simple en la que todos los objetos compartían una interfaz de gestión muy simple. A continuación se comenta los pros y contras de cada aproximación. Más adelante también se comenta el uso de JIDM-IT en el Repartidor de Operaciones de gestión.

La característica más sobresaliente de la interfaz de gestión primitiva era su sencillez. Esto redundaba en una reducción del número de ficheros a compilar: Sólo era necesario compilar con la herramienta `idl` una única interfaz, e implementarla después para los distintos componentes, sin más complicación.

Sin embargo, como también se ha comentado, su sencillez impedía el manejo de variables contenidas en tablas de una forma independiente. El uso de las Traducciones de JIDM supuso una mejora en el manejo de dichas variables, puesto que se hizo posible su acceso de manera independiente.

Por otro lado, usar la Traducción de Especificaciones de JIDM también supone varias cosas: Por un lado, cada interfaz de gestión de cada objeto contiene los atributos que mantiene como variables de gestión, lo que conceptualmente supone el saber qué variables maneja cada objeto. Sin embargo, esta mejora también supone que el módulo IDL de gestión a compilar con la herramienta `idl` posea tantas interfaces como grupos y entradas de tablas haya en el módulo SMI traducido. Además, también se debe compilar todos aquellos módulos IDL, adicionales de JIDM, que definen tipos de datos contenidos en las RFCs de definición de la MIB de Internet [Rose90] [Case90] [McCloghrie91b], además de los módulos que definen los métodos a implementar para la Traducción de Interacciones. En conjunto, el número de clases Java generadas por `idl` debidas a JIDM supera el millar, frente a la decena que se generaba con la interfaz primitiva. Este dato supone un aumento considerable del tiempo de compilación, si bien no debe resultar importante, dado que no hay por qué recompilar las interfaces de la Traducción de Interacciones y tampoco el módulo IDL de gestión basado en la MIB SMI, una vez ha sido definido.

En lo que se refiere al Repartidor de Operaciones de Gestión, también hubo cambios sustanciales tras usar JIDM. El Repartidor que utilizaba la interfaz primitiva se basaba, para la búsqueda de los objetos, en un nombrado de los objetos servidores CORBA sujeto al nombre de la MIB, lo que suponía una falta de flexibilidad, pero a la vez, evitaba tener interacciones con otros objetos distintos de aquel que poseía la información de gestión. El uso de JIDM supuso un cambio en este sentido. Se estandarizó el sistema de nombrado de objetos haciendo uso del servicio de nombres de COSS [OMG97d]. Con ello se mejoraba el criterio de búsqueda de las referencias a objetos, pero a la vez suponía que el número de interacciones se multiplicaba por dos: Primero había que interactuar con el servicio de nombres y después con el objeto gestionable en cuestión, lo que supuso una disminución en el rendimiento a cambio de flexibilidad para buscar las referencias a objetos.

3.2.2. Políticas de monitorización

Otro tema tratado en el capítulo de Diseño fue las distintas políticas de monitorización que se podían usar para visualizar el estado del sistema. A continuación se realiza, mediante

una serie de tablas, una comparativa del número aproximado de interacciones (envío de PDUs) necesarias para cada tipo de monitorización, para una situación estable del sistema.

Tabla 2. ***Monitorización SNMP desde la consola de gestión. Tres retransmisiones***

	Dominio SNMP		Dominio CORBA	Total
Número de componentes	Número de Columnas	Número de Retransmi-siones	Invocaciones (llamada y respuesta)	12 en D. SNMP
				2 en D. CORBA
14	4	3	2	14·14=196

Tabla 3. ***Monitorización SNMP desde la consola de gestión. Sin retransmisiones***

	Dominio SNMP		Dominio CORBA	Total
Número de componentes	Número de Columnas	Número de Retransmisiones	Invocaciones (llamada y respuesta)	4 en D. SNMP
				2 en D. CORBA
14	4	1	2	6·14=84

Tabla 4. ***Monitorización basada en Notificaciones***

	Dominio CORBA	Dominio SNMP	Total
Número de Componentes	Conexión con el BSM y envío de Notificación	Envío de *Trap*	3 en el D. CORBA
			1 en el D. SNMP
14	3	1	4·14=56

Tabla 5. ***Monitorización CORBA***

	Dominio CORBA	Dominio SNMP	Total
Número de componentes	Llamada al demonio y respuesta	*Traps* indicando que el BSM está lanzado, más alguno de otro componente.	2 en el D. CORBA
			1 en el D. SNMP
Indiferente.	2	2	4

En la siguiente figura se puede ver de forma gráfica la disminución de interacciones que supone el uso de una u otra monitorización.

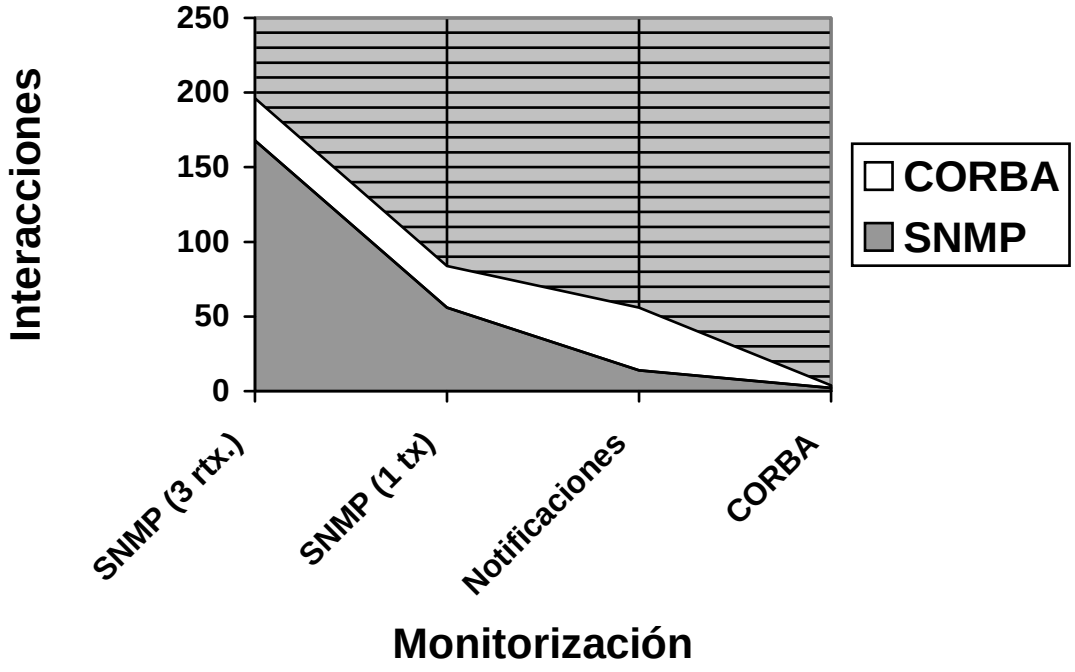


Figura 53. ***Comparativa de las distintas políticas de monitorización***

El área total de la figura indica el número de transmisiones efectuadas, separándose en transmisiones CORBA y SNMP. Si bien en el primer caso se puede reducir el número de interacciones disminuyendo el número de retransmisiones (pasando de 3 a 1), aproximándolo al valor del tercer caso, el último muestra de manera aplastante la utilidad de realizar la monitorización en el dominio CORBA. El único problema que esto supone es la mayor complejidad de manejar la monitorización desde este dominio.

3.3. Conclusiones

En este proyecto se ha presentado la arquitectura del sistema de gestión de la aplicación de intermediación de ABS como un ejemplo de cómo una aplicación basada en CORBA se puede gestionar por medio de herramientas y plataformas de gestión SNMP existentes. Los principios de JIDM han jugado un papel clave en el diseño de este sistema de gestión, principalmente en la fase de traducción de modelos de información de gestión. Las ideas de JIDM-IT se han aplicado de manera simplificada para reducir la complejidad y problemas potenciales de rendimiento en el desarrollo de la pasarela SNMP-CORBA.

Durante la integración del sistema de gestión de ABS con el prototipo del sistema de intermediación de ABS, se han obtenido las siguientes conclusiones:

1. La forma en que se ha instrumentado la información de gestión ha resultado satisfactoria, pues los desarrolladores de los componentes de la aplicación de ABS no han tenido que preocuparse de la parte de gestión del sistema. Así mismo, el sistema de gestión no se ha implementado desde cero. Se ha reutilizado la mayor cantidad de componentes existentes de otros sistemas de gestión cuando ha posible. Prueba de ello es parte de la consola de gestión, el agente extensible y el traductor de SMI a IDL.
2. El operador del sistema es capaz de obtener información de todos los entes que tengan implicaciones en ABS de forma conjunta, integrando la gestión de la aplicación con aspectos de la gestión de la red y el sistema.
3. Se puede aumentar el rendimiento realizando todas las tareas de monitorización en el dominio CORBA, evitando en lo posible la sobrecarga que produce la utilización de una pasarela SNMP-CORBA, manteniendo con el gestor una comunicación asíncrona, basada en *traps*.
4. La nota negativa es que la introducción del sistema de gestión produce una degradación importante del rendimiento de la aplicación. La activación simultánea de monitorización de componentes de la aplicación y monitorización de variables de gestión producía el agotamiento de la casi totalidad de los recursos de procesamiento disponibles. Este es un resultado claro de la sobrecarga inherente del uso conjunto de CORBA y Java. Sin embargo, la obtención de valores de variables de gestión se pudo llevar a cabo sin problemas notables de rendimiento, y por ello, consiguiendo una funcionalidad limitada de gestión de la aplicación. Además de los problemas de rendimiento, el proceso de desarrollo sufrió los ya conocidos problemas con el gran número de ficheros fuente generados por los algoritmos de JIDM [Lynch98].

3.4. Futuras líneas de trabajo

Una vez realizado el trabajo expuesto, se podría continuar el trabajar en distintos puntos para mejorar la funcionalidad del sistema de gestión desarrollado:

1. Generación automática de manipuladores de gestión Java tomando como punto de origen la definición de la MIB. Para ello se deberá desarrollar un compilador de MIBs.
2. Generación automática de código utilizando alguna herramienta de modelado que soporte UML, como la utilizada en las fases de análisis y diseño [Rational98]. Para ello habría que ampliar la metodología de desarrollo empleada, incorporando diagramas de componentes UML.

3. Mejora del gestor de notificaciones para implementar el servicio de notificaciones SNMP de JIDM.
4. Separación de la funcionalidad de JIDM-IT de los componentes de la pasarela para desarrollar servicios CORBA genéricos (como se describe en [OMG98c]). Esto incluye los servicios de nombres SNMP y de repositorio de MIBs.
5. Mejora de la funcionalidad de filtrado de notificaciones, permitiendo su configuración desde la consola de gestión.
6. Mejora de la funcionalidad de registro de notificaciones, permitiendo la utilización de varios registros distintos, configurables desde la consola de gestión.
7. Utilización de los filtros de OrbixWeb para monitorizar el proceso de obtención de información por parte del intermediario, mostrando las diferentes interacciones que se van generando[Fraunhofer97].
8. Mejora de la monitorización de variables, realizándola en el dominio CORBA y enviando traps ante una modificación, para reducir la carga debida a este proceso.
9. Detección automática de la topología del sistema a gestionar.

4. Referencias

[ABS96] ABS Consortium, *Broker Business Model*, Public Deliverable D23, octubre de 1996.

[ABS97] ABS Consortium, *Architecture and System Specification*, Deliverable D31, marzo de 1997.

[ABS98] ABS Consortium, *Broker System Version 2*, Deliverable D32, abril de 1998.

[ACTS98] ACTS, *ACTS Information Window*, http://www.infowin.org

[Advent98] Advent Network Management, Inc., *Advent Agent Builder Documentation*, http://www.adventnet.com/products/agentbuilder/help/index.html, 1998.

[Asensio97] J. I. Asensio, V. Villagrá y J. I. Moreno, *Management of Distributed Information Services: Application to the Information Brokerage Service*, Proceedings of the 4th Workshop of the Open View University Association (OVUA'97), Madrid, abril de 1997

[Asensio98a] J.I. Asensio, V. Villagrá, J. I. Moreno, J. Berrocal, *An Approach to Electronic Brokerage in TINA Environments*, Fifth International Conference on Intelligence in Services and Networks (IS&N 98). Antwerp, Bélgica, mayo de 1998.

[Asensio98b] J. I. Asensio, C. A. Iglesias, A. Tardón, A. Vielba, *Gestión de Red*, en Y. Dimitriadis, F. Díaz (Editores) *Introducción Práctica a la Administración de Sistemas en Internet*, Servicio de publicaciones de la Universidad de Valladolid, 1998, pendiente de publicación.

[Baker92] F. Baker, *IP Forwarding Table MIB*, RFC 1354, Advanced Computers Communications, julio de 1992.

[Besaw90] L. Besaw, U. S. Warrier, L. LaBarre, B. D. Handspicker, *Common Management Information Services and Protocols for the Internet (CMOT and CMIP)*, Request For Comments 1189, octubre de 1990

[Case87] J. D. Case, J. Davin, M. Fedor, M. L. Schoffstall, *Simple Gateway Monitoring Protocol*, Request For Comments 1028, noviembre de 1987

[Case90] J. D. Case, M. S. Fedor, M. L. Schoffstall y J. R. Davin, *A Simple Network Management Protocol*. Request For Comments 1157, SNMP Research, Inc., 1990.

[Case96] J. Case, K. McCloghrie, M. Rose y S. Waldbusser, *Management Information Base for version 2 of the Simple Network Management Protocol (SNMPv2)*, Request For Comments 1907, enero de 1996.

[Fraunhofer97] Fraunhofer Institut Informations-und-Datenverarbeitung, *The CORBA-Assistant: monitoring of CORBA-based Applications*, White Paper, junio de 1997.

[Harmon98] P. Harmon y M. Watson, *Understanding UML. The Developer's Guide*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1998.

[IEEE84] IEEE / ANSI Std 830. *The IEEE Guide to Software Requirements Specifications*, 1984.

[Iona96] Iona Technologies, Inc., *OrbixWeb Programming Guide*, noviembre de 1996.

[ISO84] ISO 7498: *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*, Ginebra, 1984.

[ISO90] ISO. *ISO 9595. Information Processing Systems - Open Systems Interconnections - Common Management Information Service Definition*, Ginebra, 1990.

[ISO90a] ISO. *ISO 9596. Information Processing Systems - Open Systems Interconnections - Common Management Information Protocol Specification*, Ginebra, 1991.

[ISO91b] ISO/IEC 10164-x standard, *Information Technology - Open Systems Interconnection - Systems Management - Management Functions,* Ginebra, 1991

[ISO92] ISO, *ISO 10040. Information Processing Systems - Open Systems Interconnection - Systems Management Overview*, Ginebra, 1992.

[ISO93] ISO/IEC 10165-1 standard, *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 1: Management Information Model*, Ginebra, 1993.

[ISO95] Draft International Standard ISO/IEC 13234-1, *Information Technology – Open Distributed Processing – ODP Trading Function – Part I: Specification*, Ginebra, 1995.

[ISO96] ISO/IEC JTC1/SC21 X.708, *Open Distributed Management Architecture*, Ginebra, junio de 1996.

[ITUT92a] ITU-T Rec. X.701, *Information Technology – Open System Interconnection – Systems Management Overview*, 1992.

[ITUT92b] ITU-T Rec. M.3010, *Principles for a Telecommunication Management Network*, octubre de 1992.

[ITUT92c] ITU-T Recommendation X.722, ISO/IEC 10165-4:1992, *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects, 1992.*

[ITUT94] ITU-T Recommendation X.680, ISO/IEC 8824-1:1995, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation*, 1994.

[ITUT95] ITU-T Draft Rec. X.901, *Basic Reference Model of Open Distributed Processing – Part I: Overview and Guide to Use*, mayo de 1995

[Keller98] A. Keller, *Towards Interoperable Management Architectures: Making SNMP Management Platforms suitable for CORBA*, Proceedings of the 5th Openview University Association (HP-OVUA) Plenary Workshop, Rennes, Francia, 19-21 de abril de 1998.

[Lynch98] N. Lynch, K. Hyland, *Web Enabled TMN Manager for an International Trouble Ticketing Service*. Fifth International Conference on Intelligence in Services and Networks (IS&N 98). Antwerp, Bélgica, mayo de 1998.

[Macaray96] J. F. Macaray y C. Nicolas, *Programación Java*, Gestión 2000, Barcelona, junio de 1996.

[Mazumdar96] S. Mazumdar, K. Swanson, *Web Based Management – CORBA/SNMP Gateway Approach*, Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italia, 28-30 de octubre de 1996.

[Mazumdar97] S. Mazumdar, K, Swanson, H. Sarin. *WEB Based Management: CORBA/SNMP Gateway Approach*. http://nsm.research.bell-labs.com/~mazum/CorbaSnmp.

[McCloghrie91a] K. McCloghrie and M. T. Rose. *Concise MIB Definitions.* Request for Comments 1156, Hughes LAN Systems, Inc., 1991.

[McCloghrie91b] K. McCloghrie and M. T. Rose. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, Request for Comments 1213, Hughes LAN Systems, Inc., 1991.

[Microsoft98] Microsoft, *Home Page for Microsoft Component Object Model (COM) -- Active X,* http://www.eu.microsoft.com/com/default.htm, 1998.

[NMF95] Network Management Forum, *OMNIPoint Integration Architecture*, Forum TR114, 1995.

[OMG95a] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, revisión 2.0, julio de 1995

[OMG95b] Object Management Group, *CORBAServices: Common Object Services Specification,* OMG document *95-12-30*, diciembre de 1995.

[OMG97a] Object Management Group, *A Discussion of the Object Management Architecture*, enero de1997.

[OMG97b] Object Management Group, *IDL/Java mapping specification*, OMG document orbos/97-03-01, marzo de 1997.

[OMG97c] Object Management Group, *ORB Portability Joint Submision*, OMG document orbos/97-05-25, mayo de 1997.

[OMG97d] Object Management Group, *CORBAServices: Common Object Services Specification: Naming*, OMG document formal/97-12-10, diciembre de 1997.

[OMG97e] Object Management Group, *CORBAServices: Common Object Services Specification: Event Management*, OMG document formal/97-12-11, diciembre de 1997

[OMG97f] Object Management Group, *CORBAServices: Common Object Services Specification: Property*, OMG document formal/*97-12-20, diciembre de 1997.*

[OMG98a] Object Management Group, *Updated Revised Joint Notification Service Submission*, OMG document telecom/98-03-06, marzo de 1998.

[OMG98b] Object Management Group, *CORBA/TMN Interworking Final Submission – JIDM Interaction Translation*, OMG document telecom/98-05-02, mayo de 1998.

[OMG98c] Object Management Group, *CORBA/TMN Interworking Final Submission – JIDM-SNMP*, OMG document telecom/98-05-03, mayo de 1998.

[Open92] The Open Group, *ISO and Internet Management: Coexistence and Interworking. Open Group Guide*, diciembre de 1992

[Open97] The Open Group, *Inter-domain Management: Specification Translation*. Open Group Preliminary Specification P509, marzo de 1997.

[Partridge87] C. Partridge, G. Trewitt, *High-level Entity Management System (HEMS)*, Request For Comments 1021, octubre de 1987.

[Pavlov97] G. Pavlov, *From Protocol-based to Distributed Object-based Management Architectures*. Proceedings of the 4th Workshop of the Open View University Association (OVUA'97), Madrid, abril de 1997.

[Postel80] J. B. Postel, *User Datagram Protocol*, Request For Comments 768, agosto de 1980.

[Postel81a] J. Postel, *Internet Control Message Protocol,* Request for Comments 792, septiembre de 1981.

[Postel81b] J. B. Postel, *Transmission Control Protocol*, Request For Comments 793, 1981.

[Pressman96] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. McGraw-Hill, fourth edition, 1996.

[Rational98] Rational Software Corporation, *Rational Rose 98 Enterprise Edition*, https://www.rational.com/, 1998.

[Rose90] M. T. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP based internets*. Request For Comments 1155, Performance Systems International, Inc., 1990.

[Rose91a] M. Rose, K. McCloghrie, *Concise MIB Definitions*, Request for Comments 1212, Performance Systems International, marzo de 1991.

[Rose91b] M. T. Rose, *A convention for Defining Traps for use with the SNMP*, RFC 1215, Performance Systems International, marzo de 1991

[Rose93] M. T. Rose. *The Simple Book: An introduction to management of TCP/IP based internets*, Prentice Hall, 1993.

[Rosenberry92] W. Rosenberry, D. Kenney, G. Fisher. *Understanding DCE*, O'Relly & Associaties, 1992

[Schade96] A. Schade, P. Trommler, M. Kaisersweth, *Object Instrumentation for Distributed Applications Management*, Proceedings of the IFIP/IEEE International Conference on Distributed Platforms (ICDP'96), Dresden, Alemania, 1996.

[Schmidt97] D. C. Schmidt, *Overview of CORBA*, http://www.cs.wustl.edu/~schmidt/corba-overview.html, 1997

[Schönwälder95] J. Schönwälder, H. Langendörfer. *Tcl Extensions for Network Management Applications*, Third Tcl/Tk WorkShop, Toronto, Canada, julio de 1995

[Scriptics98] Scriptics Corporation, *BLT*, http://www.tcltk.com/blt/, 1998.

[Sloman95] Ed. M. Sloman, *Network and Distributed Systems Management*, Addison-Wesley, 1995.

[Stallings93] W. Stallings, *SNMP, SNMPv2, and CMIP: The Practical Guide to Network Management Standards*, Addison-Wesley, 1993.

[Sun88] Sun Microsystems, Inc., *RPC: Remote Procedure Call Protocol Specification Version 2,* Request For Comments 1057, junio de 1988.

[Sun98] Sun Microsystems, Inc., *Java Remote Method Invocation - Distributed Computing for Java,* http://java.sun.com/marketing/collateral/javarmi.html, 1998.

[SunSoft94] SunSoft, *MIB for the SunOS SNMP Agent included with SunNet Manager 2.2.3*, 1994.

[TINA94] TINA Consortium, *Management Architecture*, Versión 2.0, diciembre de 1994

[TINA95] TINA Consortium, *Overall Concepts and Principles of TINA,* Deliverable, febrero de 1995.

[Tothenzan97] I. Tothenzan, E. Athanassiou, P. Alzon y G. T. Karetsos. *Enterprise Modelling of Information Brokerage and Retailer Services*, First International Enterprise Distributed Object Computing Workshop (EDOC'97). Marriott Resort, Gold Coast, Australia, octubre de 1997.

[Vinoski93] S. Vinoski, *Distributed Object Computing with CORBA, C++ Report*, IEEE Communications Magazine, julio/agosto de 1993.

[Vinoski97] S. Vinoski, *CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments*, IEEE Communications Magazine, febrero de 1997.

[Welch95] B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall, California, 1995.

Planos

Índice

Apéndice A. Manual de Usuario de la Consola de Gestión

Índice de figuras

Índice de cuadros

A.1. Introducción

A.1.1 Propósito

El presente manual contiene una introducción al uso del Sistema de Gestión de ABS (*Architecture for Information Brokerage Service*), una aplicación con la que se puede monitorizar y controlar el servicio ABS y su entorno.

A.1.2 Audiencia

El manual está dirigido a cualquier usuario que utilice por primera vez el Sistema desarrollado o a cualquiera que quiera conocer las características y posibilidades que ofrece el sistema. Se describirá, por tanto, el funcionamiento y las posibilidades de la aplicación ilustrándolo con ejemplos prácticos.

Se suponen unos conocimientos mínimos de informática, del sistema operativo en el que se ejecuta la aplicación y del entorno de ventanas utilizado para visualizar la interfaz de usuario.

A.1.3 Aplicabilidad

El presente manual hace referencia a la versión 2 del desarrollo del Sistema de Gestión. La aplicación deberá ser previamente instalada en una estación de trabajo con las siguientes características:

- Estación Sun SPARC (se recomienda que sea una SPARC ULTRA 1 con 64 MBytes de memoria RAM, o superior)
- Sistema Operativo SOLARIS 2.5.1
- Disco duro: 20 MBytes

A.1.4 Panorámica del documento

En el resto del documento se puede encontrar información para realizar las distintas operaciones de las que dispone la Sistema:

- En la sección A.2 se puede ver algunos conceptos que se usan en el documento y que conviene conocer.
- En la sección A.3 se incluyen los pasos necesarios para realizar las tareas básicas de la aplicación: Lanzar monitorizaciones del sistema, de variables...
- En la sección A.4 están los mensajes de error más típicos.

A.1.5 Documentos relacionados

El paquete de software del Sistema de Gestión incluye instrucciones detalladas sobre la forma en que debe ser instalada. (Véanse los ficheros `readme`).

Se puede encontrara una descripción en detalle de la arquitectura del Sistema de Intermediación ABS, su implementación y como se gestiona en las distribuciones D32 y D42 del proyecto ABS.

A.1.6 Convenciones

En esta sección se tratará de aclarar las convenciones que hemos usado en la redacción de este manual:

Se emplea la letra *cursiva* para aquellas palabras que están escritas en Inglés.

En letra `courier` se encuentran aquellas palabras asociadas a un comando o mensajes del sistema.

En letra **negrita** se encuentran aquellas palabras asociadas a un menú.

A.1.7 Notificación de erratas

Cualquier tipo de sugerencia o comentario sobre el programa o sobre el manual será bien recibido por los autores. Así mismo, cualquier error detectado puede comunicarlo por correo electrónico a las siguientes direcciones:

- Juan Ignacio Asensio Pérez <jasensio@dit.upm.es>
- Jorge Enrique López de Vergara Méndez <jlopez@dit.upm.es>

A.2. Conceptos

En este punto se explicarán la arquitectura y las funcionalidades del Gestor del Sistema de Intermediación (BSM, *Broker System Manager*) de ABS, de forma que se pueda entender el contexto en el que el Sistema de Gestión descrito en este Manual del Usuario realiza sus tareas.

El BSM-ABS trata de monitorizar y controlar los recursos empleados en la provisión del servicio de intermediación ABS. Este Sistema de Gestión se ha diseñado con la idea de soportar las cinco áreas funcionales incluidas el modelo FCAPS del modelo OSI-SM de gestión de red: fallos, configuración, anotación, rendimiento y seguridad.

El BSM-ABS adopta un paradigma de integración basado en la unificación de la red, el sistema, la aplicación y aspectos de servicios de gestión. En otras palabras, en un mismo sistema de gestión —el que está siendo descrito en este Manual del Usuario— todos los aspectos de gestión pueden ser controlados.

Para poder conseguir esa integración, y con la meta de reutilizar tantos componentes existentes como sea posible, se han elegido los estándares de gestión de Internet (los llamados SNMP, *Simple Network Management Protocol*) como base para el sistema de gestión.

Dado que el soporte para las aplicaciones del servicio de intermediación de ABS (el *Broker System*) ha sido desarrollado sobre una plataforma CORBA [OMG95], la pasarela de SNMP a CORBA es un aspecto clave del sistema de gestión.

Como se puede apreciar en la figura A.1, el sistema de gestión se compone de las siguientes partes:

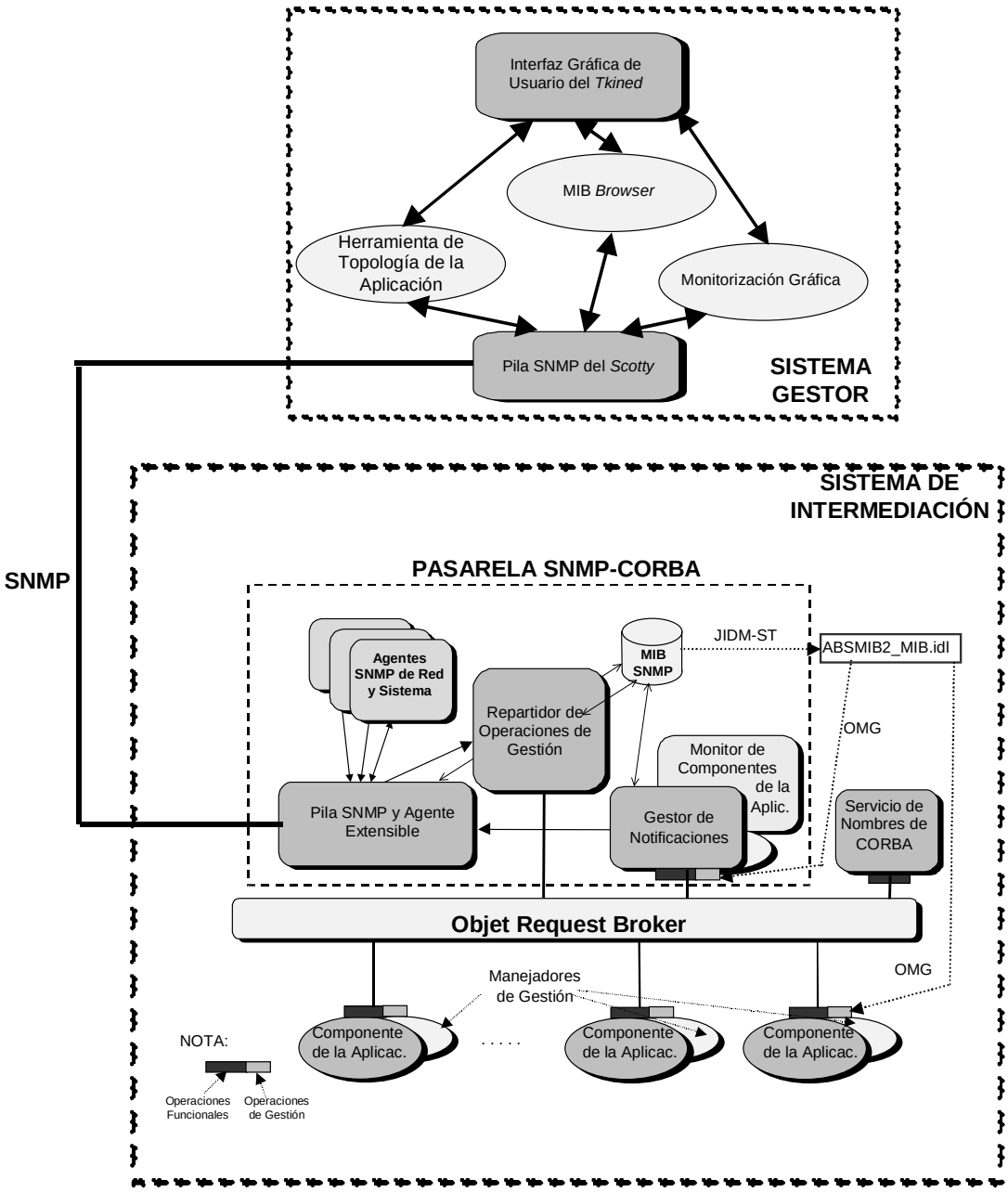


Figura A.1. ***Arquitectura del Sistema de Gestión***

- **El Sistema Gestor** (en la figura, en la parte alta): es el punto de contacto del administrador con las operaciones de gestión. Se basa en una implementación de la pila SNMP en Tcl/Tk [Welch95] de acceso público, *scotty* [Schönwälder95], que incluye un editor gráfico de redes llamado *tkined*. Mediante la adaptación de varias de las herramientas que ofrece el paquete *scotty*, y creando otras nuevas, se pueden monitorizar y controlar varios aspectos del servicio y aplicación de ABS. Para más detalles sobre el uso de este Sistema de Gestión, lea detenidamente la sección 3 de este Manual de Usuario.

- **La pila SNMP y el Agente Extensible de *Advent*** [Advent98]: Este componente (en la parte izquierda de la figura 2.1) recibe las peticiones SNMP (del Sistema de Gestión o de cualquier otro gestor de SNMP) y haciendo uso de la información de configuración, las redirige a otros componentes software específicos capaces de procesarlas. La meta es la separación de peticiones entre redes, sistemas, aplicaciones y gestión de servicio.

  Se ha usado para este componente el agente extensible desarrollado por Advent Network Management Inc. Se eligió debido a que estaba escrito en Java [Macaray96] (lo que facilitaba su integración con OrbixWeb [Iona96]) y porque es muy fácil de configurar. La información de configuración se especifica, en forma de comentario, en la MIB, que describe el modelo de información de gestión. Mediante esta información de configuración, el agente extensible sabe qué componente (un agente SNMP corriendo en otro puerto, un objeto Java, etc.) es el que debe responder a la petición de una variable de gestión específica.

  Para el Sistema de Gestión de ABS, si las peticiones SNMP se refieren a la red o al sistema de gestión, dichas peticiones se reenvían a un agente SNMP instalado en la máquina del Sistema de Intermediación. Actualmente, se usa el agente Solaris 2 SNMP que viene con la Plataforma de Gestión *SunNet Manager* [SunSoft94].

  Si la petición SNMP hace referencia a cualquier servicio o aspecto de la gestión de la aplicación de ABS, se redirigen al bloque Repartidor de Operaciones de Gestión y Pasarela, que se describe a continuación.

- **Repartidor de las Operaciones de Gestión y Pasarela**: Este componente (en la parte baja de la figura 2.1 recibe las peticiones de gestión del Agente Extensible, localiza el componente del Sistema Intermediador que puede satisfacer lo demandado, redirigir las peticiones como operaciones CORBA, recoger los resultados y devolverlos al Agente Extensible adecuando su forma para la generación de respuestas SNMP que puedan ser devueltas al Sistema Gestor.

  El Repartidor de Operaciones de Gestión y Pasarela se compone de tres clases Java que se ejemplarizan por el Agente Extensible cuando se necesita manejar una petición SNMP particular, de acuerdo con su configuración. Una de las clases se encarga de manejar variables simples de gestión SNMP, otra trata con tablas SNMP y la tercera

mantiene una tabla SNMP que almacena el estado de cada componente de la aplicación del Sistema de Intermediación (esta tabla se usa en la Gestión de la Aplicación del Sistema de Gestión).

Estas tres clases Java asumen que todos los componentes CORBA del Sistema de Intermediación implementan una interfaz IDL especial que permite la obtención y cambio de los valores de las variables de gestión. Como esta interfaz es el mismo para todos los componentes del Sistema de Intermediación, se han creado un conjunto de clases Java llamadas *Management Handlers* o manipuladores de gestión (una para cada componente CORBA del Sistema de Intermediación). Estos *Management Handlers* implementan la interfaz IDL de gestión y se usan como clases base desde las cuales se derivan los objetos CORBA del Sistema Intermediador. De esta forma, la implementación de la parte de gestión en cada componente de la aplicación está casi separada del desarrollo del mismo componente.

- **Gestor de Notificaciones**: Es un objeto CORBA que implementa una interfaz a la que se puede acceder por componentes del Sistema Intermediador para indicar que algún suceso inesperado ha ocurrido. Cuando el Gestor de Notificaciones recibe este tipo de invocación, genera una *trap* SNMP a través de la pila SNMP de Advent y la envía al correspondiente Sistema Gestor. Así mismo, mantiene un registro de las Notificaciones.

A.3. Empezando...

En esta sección se discutirán los pasos necesarios para ejecutar las distintas funciones posibles de la Sistema.

A.3.1 Materiales

Para poder ejecutar la aplicación se necesita:

- Disponer de al menos una máquina con las características y el sistema operativo que se indicó en la sección A.1.3
- Tener instalado el software del Sistema de Gestión. Normalmente la instalación y mantenimiento del software es responsabilidad del administrador de la máquina. Para ello lea los ficheros `README` que se incluyen en el paquete.

A.3.2 Preparativos

Antes de poder lanzar la aplicación, es necesario, para que se pueda hacer uso de la gestión:

- Tener lanzado el agente extensible, que se encarga de redirigir las peticiones SNMP a los distintos componentes del sistema. Para lanzarlo, es necesario tenerlo instalado previamente. Consúltese para ello los ficheros `README` que se incluyen en el paquete.

```
%cd BSM-1.3
  Se va al directorio donde se encuentra launchagent

%run_gateway
  ...Y se lanza. (Así de fácil)
```

Cuadro A.1. ***Ejemplo de Lanzamiento***

- Tener registrado en el *Repository* el gestor de Notificaciones (`BSMsrv`). En el caso de que no esté lanzado, el sistema no podrá recibir ninguna *interrupción*, lo cual implica que el sistema de gestión no será lo útil que sería en otro caso.

```
% /usr/local/bin/putit -j –classpath
${ABSHOME}/classes:.:/opt/OrbixWeb/classes:/opt/JDK/lib/classes.zip BSMSrv
BrokerSystem.ServiceSession.BSM.BSMserver <nombre de máquina> <fichero MIB>
<fichero de registro>
```

Se registra el componente, diciendo en que máquina está la consola de gestión y el nombre del fichero que contiene la Management Information Base

```
%/usr/local/bin/chmodit i+all BSMSrv
%/usr/local/bin/chmodit l+all BSMSrv
```

...Y se hace accesible por cualquiera.

Cuadro A.2. ***Ejemplo de registro del BSM***

- Tener registrado el resto de los componentes. En el caso de que no haya registrado ninguno, no será posible acceder a ellos, y por tanto, administrarlos.

A.3.3 Precauciones y advertencias

- Si no se realizan los preparativos del apartado anterior, el sistema puede que dé algún error y no será posible gestionar los componentes.
- Deberán estar registrados aquellos componentes que se quieran gestionar, no siendo necesario para aquellos que no se desee. En todo caso debe estar lanzado el agente.

A.3.4 Método

En este apartado se tratará de aclarar cómo utilizar la aplicación, paso a paso.

A.3.4.1 Lanzar la aplicación

Una vez han sido realizados todos los preparativos, es posible lanzar la aplicación. Veamos un ejemplo:

```
% cd manager
```

Se va al directorio donde está el script de lanzamiento

```
% run_manager
```

Se ejecuta el script. Se preguntará si quiere lanzar la aplicación en una ventana aparte.

*Responda y ó n según desee. Finalmente, si todo ha ido bien, se visualizará la ventana principal de la aplicación de gestión (*figura A.2*)*
En el caso de que `run_manager` *no tenga permisos de ejecución, se deberá ejecutar:*
```
% csh run_manager
```

Cuadro A.3. ***Ejemplo de lanzamiento de la aplicación.***

A.3.4.2 La ventana principal

Una vez ha sido arrancada la aplicación y se visualiza la ventana de la figura 3.1, ya se puede empezar la administración del sistema, aunque previamente se va a dar una descripción de dicha ventana:

- En la parte más alta está la barra con información sobre la ventana, así como los botones de maximizar, minimizar y destruir.
- En la parte alta de la ventana se encuentra la barra de menús, entre los que se encuentran **File**, **ABS** y **Help**.
- En el lado derecho y en la parte baja se encuentran las barras de desplazamiento que sirven para desplazar la imagen.
- En el centro de la ventana se encuentra la imagen que representa el mapa de Europa, y el conjunto de subsistemas de ABS:
    - En color amarillo, los proveedores de contenido (CP), sobre los que no se tiene ninguna posibilidad de gestión
    - En color azul, los proveedores de servicio (SP), sobre los que se puede hacer gestión de la aplicación

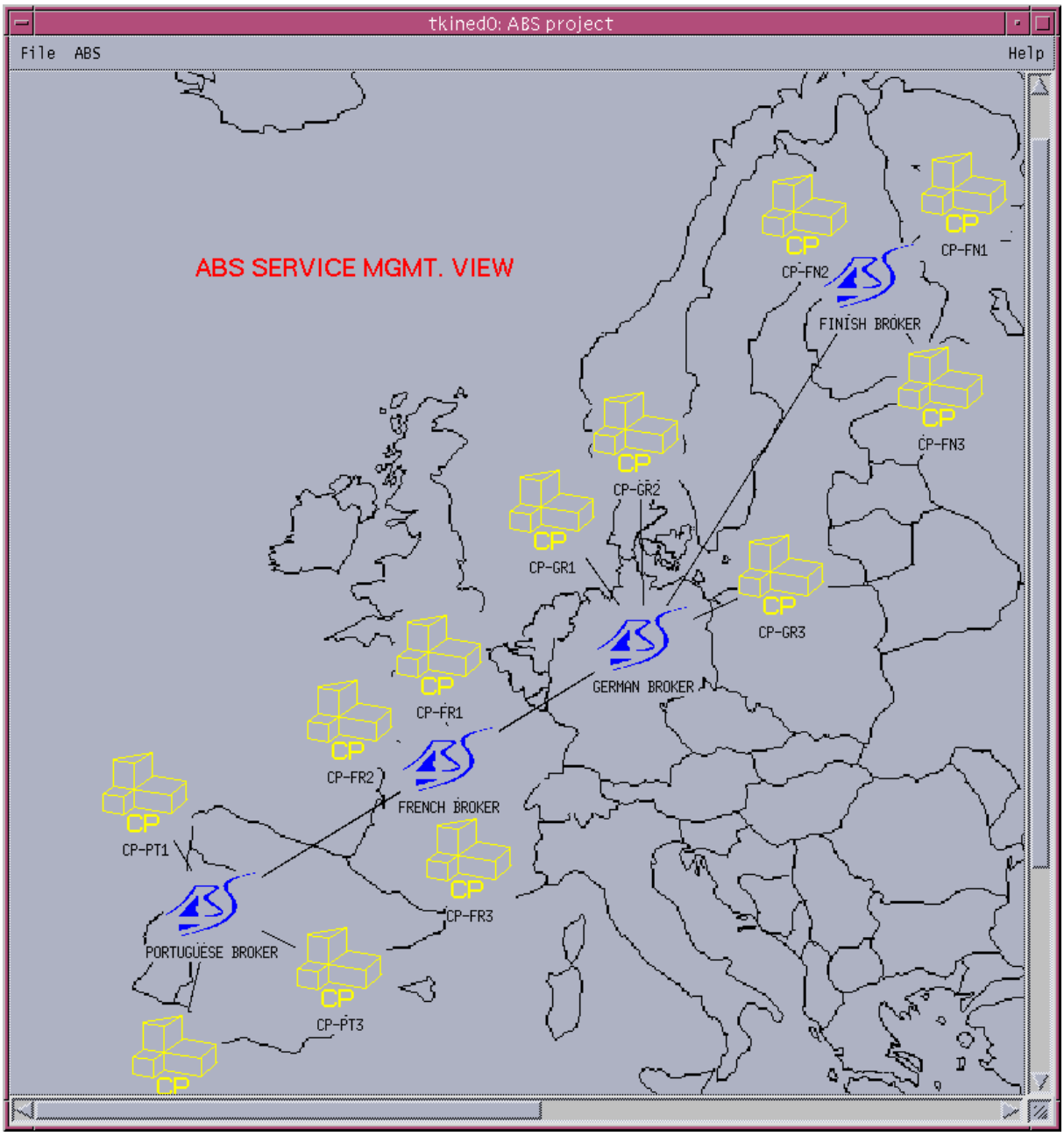


Figura A.2. ***Ventana Principal de la Aplicación***

Las funciones que se pueden encontrar en los distintos menús son:

- Menú **File**: Contiene los siguientes comandos:
  - **Print**: Como su nombre indica, genera un fichero Postcript de la pantalla para ser impreso.
  - **Close View**: Sirve para cerrar la aplicación. Al ejecutarlo nos pide confirmación.

- Menú **ABS**: Posee las siguientes funciones:
    - **Application Management**: Despliega una nueva ventana con los distintos componentes que posee un nodo
    - **SNMP Tree**: Visualiza una ventana con el árbol MIB de ABS. Si no se selecciona ningún objeto, no se podrá hacer uso de gran parte de sus funcionalidades, puesto que no se sabrá a qué dirección IP hay que hacer las peticiones.
    - **Help ABS Menu**: Visualiza sucintamente las funcionalidades de este menú.
- Menú **Help**: Es un menú propio del Tkined1.4.5 (software sobre el que está implementada la Sistema), y da una somera ayuda del entorno.
- Menús **Desplegables**: Son los menús que aparecen al apretar el botón derecho del ratón cuando el puntero está sobre alguno de los objetos (CP y SP).
    - En general tienen estas funciones:
        - **Create a new attribute...**: Como su nombre indica, sirve para añadir un nuevo atributo a los que ya posee dicho objeto
        - **Delete an attribute...**: Es la función contraria a la anterior: Borra uno de los atributos del objeto
        - **Edit all attributes...**: Es la función que le será de más utilidad al usuario, pues sirve para editar los atributos del objeto, entre los que se encuentra la dirección IP, que deberá ser cambiada a un valor adecuado.
        - **Label with attribute...**: Sirve para cambiar la etiqueta del objeto haciendo uso de uno de los atributos que posee
    - Los SP tienen además:
        - **Application Management**: Es la misma función que se encuentra en el menú ABS
        - **Network Management**: Esta función no está actualmente

implementada.

- **SNMP Tree**: Esta función es la misma que la que se encuentra en el menú ABS. En este caso, se visualiza el árbol MIB del objeto sobre el que se ha pulsado.

Para finalizar este punto, se recomienda al lector que se familiarice con los distintos componentes a los que se ha hecho mención en este apartado.

A.3.4.3 Primeros pasos

En esta sección se pretende que el usuario aprenda a lanzar la ventana de **Gestión de la Aplicación**. Dicha ventana permite gestionar los distintos componentes de la aplicación: Hacer monitorizaciones del estado de los componentes, de sus variables de gestión, de las *traps* que llegan...

Siga estos pasos:

1. Primeramente tiene que colocar los valores correctos correspondientes a su SP:
    1. Ponga el puntero del ratón sobre el SP sobre el que va a realizar la administración.
    2. Pulse y mantenga el botón derecho del ratón. Debe salirle un Menú Desplegable. Si no es así, es que no tiene bien posicionado el puntero.
    3. Escoja el comando **Edit all attributes**... Debe aparecer una ventana con todos los atributos del objeto.
    4. Edite aquellos que considere conveniente. Por ejemplo, escriba la dirección IP correcta de la máquina donde se encuentra corriendo el agente.
1. Ahora que ya están todos los datos correctamente, pruebe a lanzar la Gestión de aplicación. Tiene dos métodos posibles:
    1. El primero es mediante la Barra de Menús:
        1. Pulse con el botón izquierdo del ratón el SP a gestionar. Deberán aparecer cuatro puntos cuadrados alrededor de dicho nodo. Si no es así, vuelva a intentarlo.

2. Seguidamente vaya al menú **ABS** y escoja la función **Application Manager**. Con ello se visualizará una nueva ventana, como la de la figura 3.2, que contiene el mapa de los distintos componentes de la aplicación (componentes del Sistema de Intermediación de ABS, BSC (*Broker System Component*))

2. El segundo método hace uso del Menú desplegable:

    1. Pulse y mantenga el botón derecho del ratón sobre el SP a gestionar. Debe salirle un Menú Desplegable.

    2. Escoja la función **Application Manager**. Con ello se visualizará una nueva ventana, como la de la figura 3.2, que contiene el mapa de los distintos componentes de la aplicación (BSC)

2. Con estos pasos ya está preparado para empezar a hacer monitorizaciones.

Cuadro A.4. ***Primer ejemplo de la Aplicación***

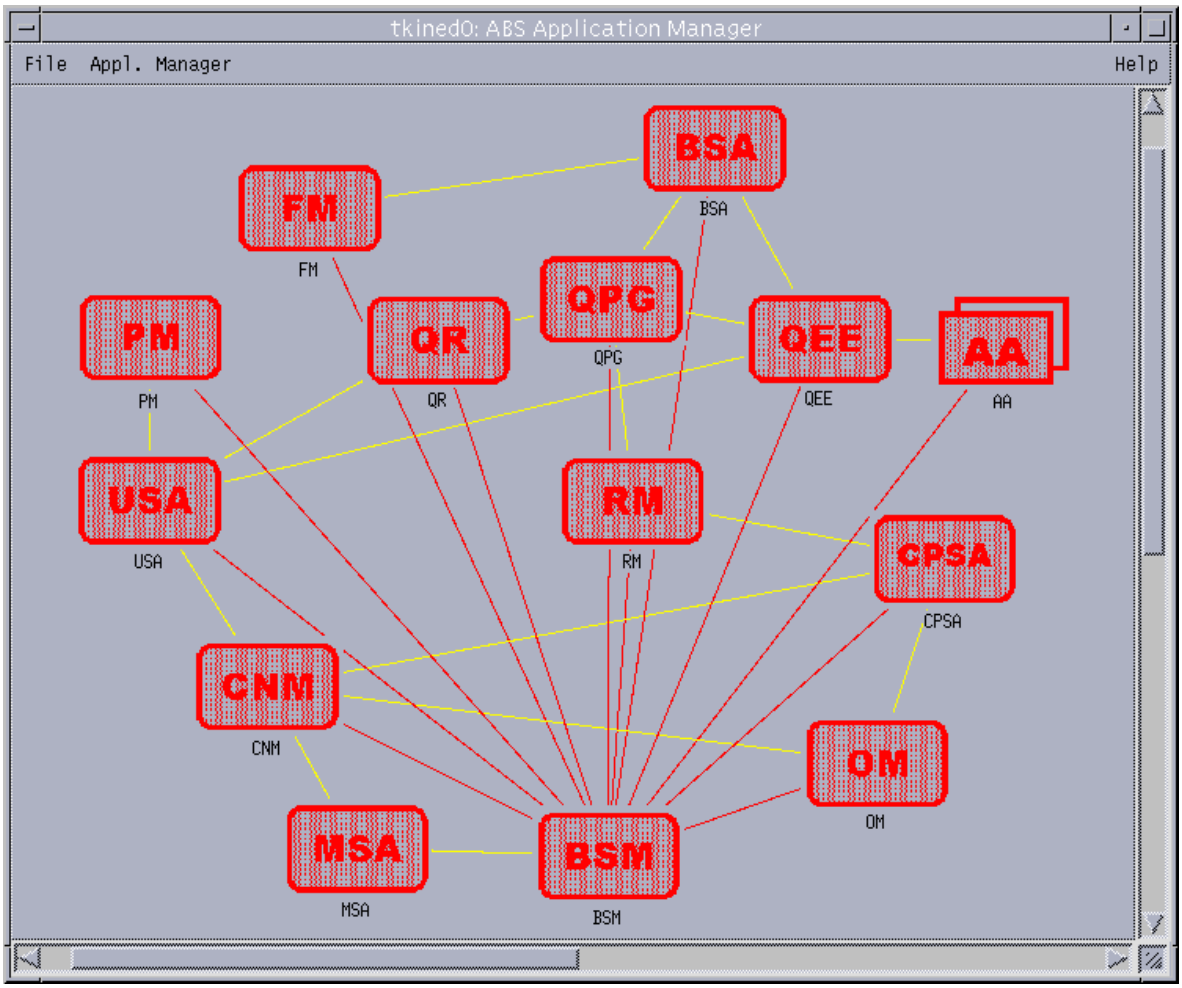


Figura A.3. ***Ventana de Gestión de la Aplicación***

A.3.4.4 Gestión de la Aplicación. Monitorización de estados

La ventana de Gestión de la Aplicación (figura 3.2.) es bastante similar a la ventana principal. De hecho, únicamente cambia el mapa visualizado y que en vez del menú **ABS** está el menú **Appl. Manager**. También se visualiza una ventana de información como la de la figura 3.3, pero que no añade funcionalidad al conjunto.

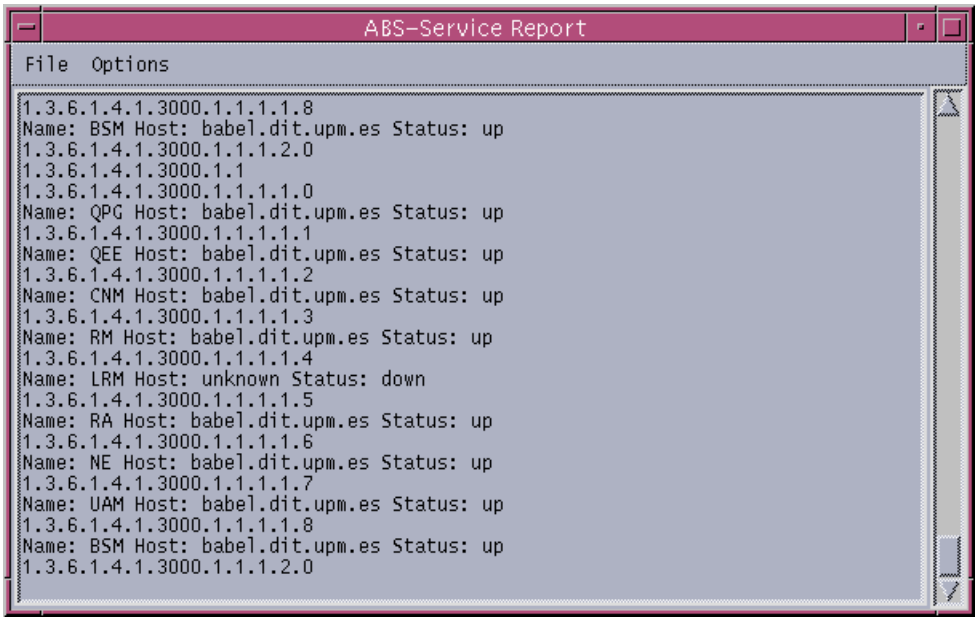


Figura A.4. ***Ventana de Registro de Información***

El menú **Appl. Manager** posee las siguientes funciones:

- **Start Monitor**: Comienza a realizar la monitorización de estados desde el gestor SNMP.
- **Stop Monitor**: Para la monitorización de estados desde el gestor SNMP.
- **TTL Monitor**: Comienza la monitorización basada en notificaciones.
- **BSM Monitor**: Comienza la monitorización desde el dominio CORBA.
- **Stop TTL**: Para las monitorizaciones TTL y BSM.
- **Nice Monitor**: Comienza una monitorización combinada SNMP-CORBA.
- **Stop Nice**: Para la monitorización combinada.
- **Trap Monitor**: Habilita e interrumpe la recepción de interrupciones SNMP.
- **SNMP Tree**: Visualiza un árbol basado en la MIB de ABS. La función es similar a la del menú **ABS,** sólo que en este caso, si hay algún componente seleccionado, se visualiza sólo la MIB de dicho componente.
- **Set SNMP variables**: Sirve para modificar las variables del protocolo de gestión SNMP

- **Help Appl. Manager**: Lanza una ventana que explica más o menos lo que se cuenta en estos puntos.

En esta ventana, los **menús desplegables** son como los ya comentados en el apartado 3.4.2, e incluyen la función **SNMP Tree**, relativa a cada componente.

Una vez se han contado los puntos esenciales relativos a esta ventana, se puede ver el segundo ejemplo de la aplicación, relativo a la monitorización de estados o simplemente monitorización. Se recomienda que, como en ocasiones anteriores, el usuario se familiarice previamente con los nuevos menús a los que tiene acceso.

Una vez se ha hecho el primer ejemplo y se encuentra la ventana ABS Application Manager abierta, ejecute los siguientes pasos:

1. Vaya al menú **Appl. Manager**
2. Seleccione **Nice Monitor**
3. Aparecerá una ventana de diálogo en la que se solicita el tiempo entre monitorizaciones. Introduzca un nuevo tiempo o simplemente pulse el botón de aceptar.

La ventana de información empezará a visualizar mensajes de estado de cada componente, a la vez que los componentes pasarán a verde si están lanzados, o se quedarán en rojo si no lo están.
Si la ventana de información da mensajes de error es que posiblemente no esté lanzado el agente.

Cuando quiera parar la monitorización:

1. Vaya al menú **Appl. Manager**
2. Seleccione **Stop Nice**

Puede que note cierta inercia en la parada de la monitorización, pero no se preocupe, es que se están recibiendo las últimas peticiones realizadas.

Cuadro A.5. ***Segundo ejemplo de la aplicación: Monitorización de estados***

A.3.4.5 SNMP Tree. Monitorización de variables

El SNMP Tree es una funcionalidad que incluye el Tkined1.4.5. Su funcionamiento consiste en que dado un OID (*Object Identifier ASN.1*, identificador de objeto), muestra todas las ramas que *cuelgan* de dicho OID. Al ser llamado aparece una ventana como la que se muestra en la figura 3.4, y su barra de menú incluye los siguientes:

- Menú **File**:
  - **Load**: Esta opción se usa para cargar un nuevo fichero que contenga una MIB en la herramienta SNMP Tree (no se usará muy frecuentemente, puesto que las MIBs más importantes, incluyendo la MIB de ABS, se cargan por defecto).
  - **Find**: Se puede usar para localizar una variable de gestión particular dentro del conjunto de MIBs cargadas.
  - **Walk**: Recupera de forma recursiva los valores de las variables de gestión que están bajo un particular nodo de una MIB.
  - **PostScript**: Genera un fichero *postscript* gráfico con el contenido de la ventana de la herramienta.
  - **New View**: Lanza una nueva ventana SNMP Tree.
  - **Close View**: Cierra la ventana. Siempre se debe usar este comando para cerrar la ventana.
- Menús **Mib-2**, **Transmission**, **SNMPv2**, y **Enterprise**: Se usan para seleccionar las distintas MIBs cargadas. Por ejemplo, la MIB ABS (La que se visualiza al arrancar la herramienta) se encuentra en el menú **Enterprise**.
- Menú **Bookmarks**: Se usa para recordar lugares del árbol de la MIB(como en un *Web Browser* típico).
- Menú **Options**: Gracias a las opciones de este menú, se puede visualizar gráficamente algunos atributos de la definición de variables de gestión.
- **Zoom in / Zoom out**: Los dos símbolos situados al lado del menú **Options** realizan la función de *zoom* del árbol que representa la información de gestión.

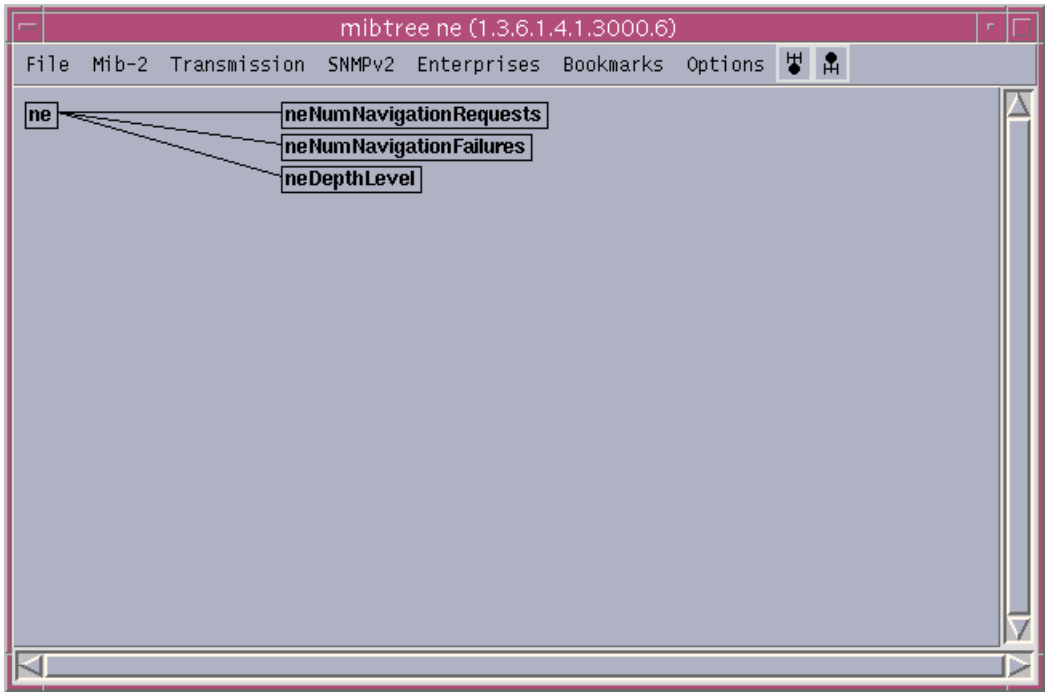


Figura A.5. ***Ventana del SNMP Tree***

Además, también hay disponibles menús desplegables, que sí se usarán, que según el objeto sea *rama* u *hoja* permite realizar alguna de las siguientes funciones:

- **Describe MIB variable**: Da información sobre las distintas variables.
- **Show MIB variable**: Realizar un *get* a una variable
- **Show MIB table**: Realiza varios *gets* sucesivos hasta que se muestra una tabla en su totalidad.
- **Walk MIB node**: Realiza una especie de *getnext*: Da los valores de todas las *hojas* que hay a partir de esa *rama*
- **Set Value**: Realiza un *set* a una variable (debe ser *read-write* para que se pueda escribir)
- **Monitor MIB variable**: Hace una monitorización gráfica de dicha variable.
- **Goto Node**: Avanza o retrocede dentro del árbol.

Para acceder a toda esta información, la monitorización de estado debe estar en funcionamiento, pues de lo contrario, SNMP Tree no puede encontrar la dirección IP donde está el agente para hacer las peticiones.

Se va a ver por último un ejemplo donde poner en práctica algo de lo que se ha dicho en este apartado.

Una vez ha realizado los dos ejemplos anteriores realice el siguiente ejemplo en los siguientes pasos:

1. Compruebe que se está realizando monitorización de estado. (La ventana de información está añadiendo líneas, es decir: "no está en reposo")
2. Lo segundo se puede hacer de dos formas:
    1. Seleccionar uno de los componentes que estén en verde, el que se quiera ver, ir al menú **Appl. Manager** y pulsar en la función **SNMP Tree**.
    2. O bien, ir directamente al componente elegido y pulsar el botón derecho. Una vez que sale el menú desplegable escoger la función **SNMP Tree**
3. Con ello, sale la ventana de la figura 3.4 y podemos empezar a familiarizarnos con el entorno.
4. El árbol tiene la *raíz* a la izquierda y las *hojas* a la derecha.
5. Vaya a una de las *hojas* que sea de tipo INTEGER (se puede saber haciendo un **Describe MIB Variable** abriendo el menú desplegable)
6. Una vez ha situado el puntero del ratón sobre dicha *hoja*, pulse el botón derecho y escoja **Monitor MIB Variable**.
7. Si todo se ha hecho correctamente, aparecerá una ventana de diálogo en que se pide la longitud de la *ventana* de monitorización y el tiempo entre peticiones, así como si se quiere que la gráfica muestre valores diferenciales (detector de eventos) o bien valores absolutos.
8. Finalmente, y al pulsar aceptar, tendremos una nueva ventana (figura 3.5)con una gráfica que nos muestra el valor de dicha variable, y que irá variando con el tiempo si dicha variable es modificada por el componente.

***Nota**: Este ejemplo está referido al SNMP Tree al que se accede desde la ventana de* Gestión de la Aplicación*, pero puede ser igualmente realizado desde la* ventana principal*,*

prescindiendo del punto 1.

Cuadro A.6. ***Tercer ejemplo de la aplicación: Monitorización de variables***

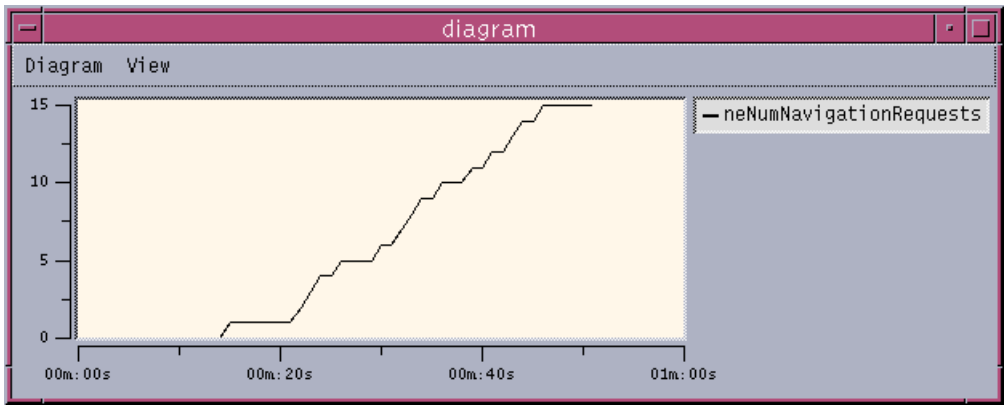


Figura A.6. ***Ventana de Monitorización de Variables***

La ventana de monitorización de variables tiene las siguientes características:

- Una barra de menús:
  - Menú **Diagram**: Tiene los siguientes comandos:
    - **Logaritmic Y Scale**: Para pasar la escala de ordenadas a logarítmica
    - **Linear Y Scale**: Para pasarla a lineal.
    - **Set Window Length**: Para cambiar la longitud de la ventana de monitorización.
    - **PostScript**: Para hacer un volcado del gráfico a un fichero *PostScript*
  - Menú **View**: Tiene una única función, que es cerrar la ventana (**Close view**)
- Una zona gráfica donde se visualiza la variación de la variable. Cuando se supera el rango de abscisas, este se va desplazando hacia la izquierda.

A.3.4.6 Monitorización de Traps

Otra funcionalidad que incluye el Tkined1.4.5 y que se ha adaptado al tipo de información que se utiliza en ABS es un receptor de traps. En este último caso, una vez llegan

traps de un componente, este se pone a parpadear en un color determinado según sea la trap:

- **Azul claro** para la interrupción 1: *unknownManagementInformation*
- **Rojo** para la interrupción 2: *genericSoftwareFailure*
- **Magenta** para la interrupción 3: *setVariableNotAllowed*
- **Verde** para la interrupción 4: *componentIsReady*
- Y **negro** cuando la interrupción es desconocida.
- Además, si no se reconoce el componente que envió la *trap*, se le asigna a BSM, parpadeando en este caso en **rosa**

A la vez que ocurre esto, se abre una ventana de información como la de la figura 3.6 —que salvo la información que visualiza es similar a la de la figura 3.3— que indica la trap que ha sido recibida, así como la hora y la dirección IP de procedencia.

Únicamente se atienden las traps que vienen de la máquina donde se encuentra el módulo BSM

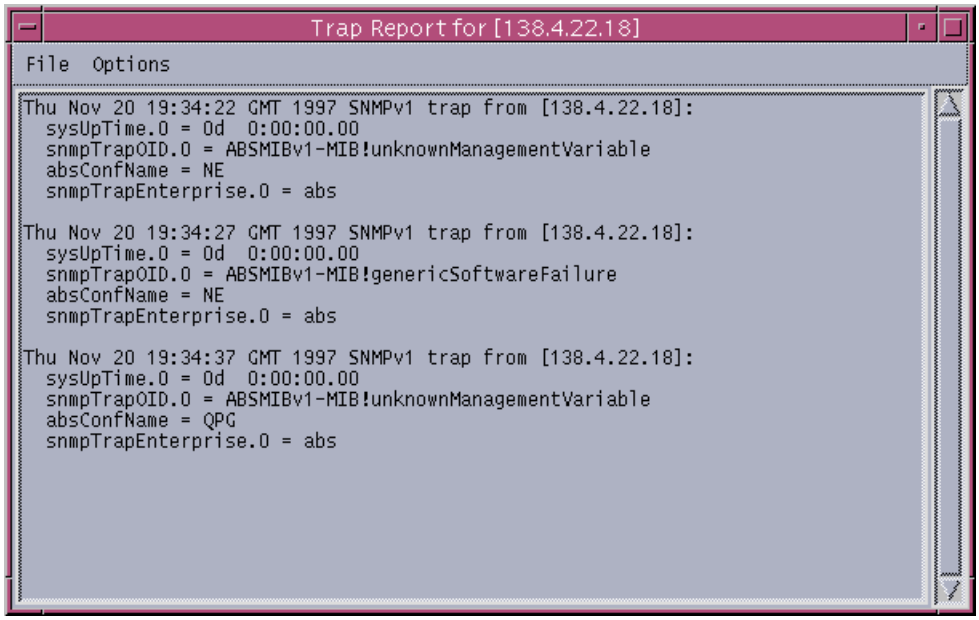


Figura A.7. ***Ventana de información y registro de*** **traps**

Dado que la recepción de traps depende externamente de los componentes, no es posible ver un ejemplo en que se reciban dichas interrupciones. Así pues, sólo se enseñará como se puede habilitar y deshabilitar su monitorización.

Para realizar el siguiente ejemplo es necesario haber realizado los dos primeros. Es independiente del tercero. Siga estos pasos:

1. Vaya al menú **Appl. Manager** y escoja el comando **Trap Monitor**.
2. Aparecerá una ventana de diálogo que le preguntará si debe escuchar (*listen*) o ignorar (*ignore*) las interrupciones SNMP que lleguen.
3. Escoja escuchar y ya está. Sólo debe esperar a que ocurra alguna.

Cuadro A.7. ***Cuarto ejemplo de la aplicación: Monitorización de*** **Traps**

A.3.4.7 Cambiando variables de configuración de SNMP

Otra funcionalidad adicional del Tkined1.4.5 es poder efectuar cambios sobre las distintas variables de SNMP. Las variables que pueden ser modificadas son:

- **Community**: Comunidad
- **UDP Port**: El puerto donde va a realizar las peticiones
- **Timeout**: El tiempo que va a esperar a una respuesta
- **Retries**: El número de veces que se va a tratar de obtener una respuesta
- **Window Size**: Ventana de transmisión
- **Delay**: Retardo de transmisión
- **Protocol**: Los protocolos posibles que hay: SNMPv1, SNMPv2c, SNMPv2u
- **User**: Usuario.
- **Context**: Contexto.
- **List Alias**: Da una lista de los nombres de red que poseen alias.

  A continuación se ofrece un ejemplo de como usarla.

Habiendo realizado los ejemplos uno y dos dé los siguientes pasos:

1. Vaya al menú Appl. Manager y escoja el comando **Set SNMP Variables**.

2. Aparecerá una ventana de diálogo como la de la figura 3.7 en que puede ver y cambiar las variables SNMP que se están utilizando.

3. Cambie las que cree necesario y pulse Aceptar ó bien, Cancelar, si no desea efectuar ningún cambio.

Cuadro A.8. ***Quinto ejemplo de la aplicación: Cambiando variables de configuración de SNMP***

Figura A.8. ***Ventana de configuración de variables SNMP***

A.3.4.8 Saliendo de las distintas ventanas

Para finalizar el apartado 3.4, hay que comentar como salir o cerrar las distintas ventanas. Es tan fácil como ir al menú **File** de cada una de ellas y hacer un **Close view**.

A.3.5 Información relacionada

El sistema está formado por varios objetos Java sobre una plataforma distribuida CORBA. Para registrarlos es necesario que previamente hayan sido compilados, así como tener el demonio de *orbix* (`orbixdj`) corriendo.

A.4. Errores y posibles soluciones

A.4.1.Mensajes de error

A.4.1.1. Errores que aparecen por la salida estándar

Los siguientes errores aparecen por la salida estándar, que normalmente será la pantalla, a no ser que esté redirigida a otro dispositivo. Todos producen la salida de la aplicación.

A.4.1.1.1. `MGMTHOME variable not set`

Este error se produce si la variable de entorno no ha sido definida previamente. Por alguna razón, no se ha hecho un source del `MGMTHOME` (porque no se ha ejecutado `run_manager` o porque el fichero `mgmtenv` no está o está mal...)

A.4.1.1.2. `xxx.tcl was not found. Check MGMTHOME variable`

```
Problems when creating interpreter. Check MGMTHOME variable
```

```
BLT Tcl library not found. Check MGMTHOME variable
```

Los errores que acaban en `Check MGMTHOME variable` se producen debido a que no se encuentran algunos ficheros. Esto puede ser debido a dos razones:

1. Por error ha borrado alguno de los ficheros que componen el sistema.
2. La variable `MGMTHOME` no ha sido definida previamente de manera correcta. No debe ocurrir si ha lanzado la aplicación con el comando `run_manager`.

A continuación se exponen los posibles errores que puede dar la ejecución del comando `run_manager`.

A.4.1.1.3. `MANAGER RUN: PWD variable not set`

```
MANAGER RUN: Please, substitute the PWD variable by the name
```

```
MANAGER RUN: of the working directory in the mgmtenv file
```

El sistema no tiene la variable `PWD` como variable de entorno. Créela con un `setenv`

PWD <directorio_de_run_manager>.

A.4.1.1.4. MANAGER RUN: file mgmtenv not found

MANAGER RUN: Please, obtain a copy of this file

El fichero mgmtenv no se encontró. Posiblemente fue borrado accidentalmente o está ejecutando el comando desde otro directorio. En el primer caso, obtenga una copia de dicho fichero. En el otro, vaya al directorio MANAGER-1.3 para ejecutarlo.

A.4.1.1.5 MANAGER RUN: MGMTHOME, TCL_LIBRARY, TCLLIBPATH,

MANAGER RUN: TNM_LIBRARY and TKINED_PATH variables

MANAGER RUN: must be set

El programa necesita de las variables MGMTHOME, TCL_LIBRARY, TCLLIBPATH, TNM_LIBRARY y TKINED_PATH para poder funcionar. Seguramente el fichero mgmtenv fue editado y estas variables se borraron. Configúrelas en los valores adecuados.

A.4.1.1.6 MANAGER RUN: The ${MGMTHOME} directory does not exist...

MANAGER RUN: Have you unpacked the proper version of the

MANAGER RUN: Manager and in the correct place?

Se ha tratado de acceder al directorio al que apunta la variable MGMTHOME, pero este no existía. Posiblemente no se haya instalado correctamente el programa, o bien, se haya borrado.

A.4.1.1.7 MANAGER RUN: DISPLAY variable no set

El programa se está corriendo en modo remoto y la variable de entorno DISPLAY no se ha configurado, con lo que no se puede abrir ninguna ventana. Haga un setenv DISPLAY nombre.de.mi.máquina:0.0.

A.4.1.1.8 MANAGER RUN: /usr/openwin/bin/xterm is not found...

MANAGER RUN: You might launch the manager in this terminal

No se encontró el programa xterm en la ruta especificada. Deberá lanzar el programa en esta ventana, o bien colocar un enlace a xterm en dicho directorio (ln -s /usr/openwin/bin/xterm ruta_donde_esté/xterm).

A.4.1.1.9 `MANAGER RUN: Unable to launch the appl. manager in a separate windows`

`MANAGER RUN: Check if the file ${MGMTHOME}/bin/tkabs exists`

No fue posible lanzar la aplicación. Debe comprobar si el fichero especificado existe. Si no es así, obtenga una copia. Si no, no ha sido posible su ejecución debido a problemas internos de Tkined.

A.4.1.1.10 `MANAGER RUN: Response not valid. Try again...`

Respondió algo distinto a `y` (para lanzar en una ventana distinta) o `n` (para una misma ventana). Sin duda, se ha equivocado de tecla.

A.4.1.2. Errores que aparecen en una ventana de confirmación

Este tipo de errores aparece en una ventana de confirmación. Esta ventana tiene un botón de `DISMISS` para cerrarla. La ocurrencia de uno de estos errores únicamente implica que no se ejecutará la función que se había seleccionado.

A.4.1.2.1. `Please, select just one ABS Srv. Provider`

Ha realizado la selección de más de un SP, y únicamente se puede referir a uno para abrir la ventana de Gestión de la aplicación. Realice la selección de un único SP y vuelva a intentarlo.

A.4.1.2.2. `Please, select an ABS Srv. Provider`

Ha realizado la selección de un objeto distinto a un SP, con lo que no se puede abrir la ventana de Gestión de la aplicación. Realice la selección de un SP y vuelva a intentarlo.

A.4.1.2.3. `Environment variable MGMTHOME was not set`

Este error es similar al A.4.1.1.2, y si se produce, no podrá abrir la ventana de Gestión de la aplicación. ¿Ha arrancado la aplicación con `run_manager`?

A.4.1.2.4. `Problems when creating the Appl. Manager Interpreter`

Este error se puede producir al abrir la ventana de Gestión de la aplicación. Puede ser debido a la variable `MGMTHOME`, o bien a algún error interno del Tkined. En cualquier caso, no podrá abrir la ventana de Gestión de la aplicación.

A.4.1.2.5. `ABS MIB not found`

Al cargar la MIB de ABS, esta no fue encontrada. Puede que haya que accidentalmente la haya borrado, o bien que la variable `MGMTHOME` esté mal configurada. Este error puede afectar a la hora de hacer Monitorización de estados y a la hora de visualizar el SNMP Tree de un componente.

A.4.1.2.6. `Monitor already started!`

Se ha vuelto a querer lanzar la Monitorización de estados, cuando ésta ya está en funcionamiento.

A.4.1.2.7. `Address could not be received`

Debido a este error, no se podrá realizar Monitorización alguna, pues al lanzar la ventana de Gestión de la aplicación, no se pasó correctamente la dirección IP de BSM.

A.4.1.2.8. `Please, select just one ABS Service`

Ha seleccionado más de un AS para ver su SNMP Tree. Seleccione únicamente un AS.

A.4.1.2.9. `mibtree script could not be launched`

No se encotró el fichero `mibtree`, y no podrá lanzar el SNMP Tree. Como en casos anteriores, o bien faltaba el fichero o bien la variable `MGMTHOME` era errónea.

A.4.1.2.10. `xxx node not found in the MIB`

Se ha querido visualizar el SNMP Tree de un AS que no posee información de gestión.

A.4.1.2.11. `straps port failed`

Al tratar de acceder al puerto de las *traps*, este no ha sido accesible. Vuelva a intentar la operación. Si error persiste es posible que el fichero `straps` no tenga derechos de superusuario.

A.4.1.3. Errores que aparecen en la ventana de información

Los siguientes errores se visualizan por la ventana de información y son debidos a

problemas en la comunicación de red durante la Monitorización de estados.

A.4.1.3.1. `Configuration (applmgmt.tcl) SESSION ERROR: *********`

No se pudo abrir la sesión SNMP para realizar la Monitorización de estados. Seguramente, debido a un problema interno del Scotty/Tkined, o bien de sobrecarga de la red. `*********` indica la posible causa de dicho error.

A.4.1.3.2. `Configuration (applmgmt.tcl) GET-NEXT ERROR: *********`

Se produjo un error en la sesión SNMP al realizar una operación *getnext* durante la Monitorización de estados. `*********` indica nuevamente la posible causa del error. Una posible causa, si se produce varias veces seguidas, es que el agente no esté funcionando o bien que haya *muerto.* En caso contrario se debe simplemente a que el PDU se perdió.

A.4.1.3.3. `Configuration (applmgmt.tcl) GET ERROR: *********`

En este caso, el error de la sesión SNMP se produjo al realizar un *get* durante la Monitorización de estados. `*********` nos muestra otra vez la posible causa del error. Una posible causa, si se produce varias veces seguidas, es que el agente no esté funcionando o bien que haya *muerto.* En caso contrario se debe simplemente a que el PDU se perdió.

A.4.1.3.4. `Error: premature end of table`

Al obtener la tabla de configuración de ABS para la Monitorización de estados, ésta terminó antes de tiempo. Si el sistema está estable no debe ocurrir este problema.

A.4.1.3.5. `DNS failed`

El servidor de DNS de la red no ha podido resolver una dirección IP.

A.4.1.4 Otros errores no contemplados en este manual

En el punto 4.1 se han contemplado únicamente los errores detectables a partir del código desarrollado para el proyecto ABS. Otro tipo de errores será debido a la plataforma Tkined/Scotty, por lo que se debe buscar en su documentación la posible causa de los mismos.

A.4.2. Errores conocidos

En esta sección se tratan aquellos errores de los que se tiene constancia pero que no han podido ser evitados.

A.4.2.1. Errores de Tkined

A.4.2.1.1. `Grab failed. Another application has grab`

Si la aplicación se está corriendo remotamente es posible que al abrir algún menú le aparezca este mensaje. Responda `OK` y siga trabajando. Si el menú que iba a abrir ya no abre, no se preocupe: Abra cualquier otro y tras ello vuelva a intentar abrir el primero.

A.4.2.1.2. `Invalid path .top0...`

Si cierra la ventana del SNMP Tree de forma abrupta (con el botón de destruir la ventana), le aparecerán varias ventanas con un mensaje parecido. Pulse `OK` en todas ellas y recuerde que se debe cerrar haciendo uso de **Close view** en el menú **File**.

A.4.2.2. Otros errores

En esta sección se tratará de ver algunos errores comunes que se pueden producir, sin que tenga por qué aparecer un mensaje de error específico, y su posible solución.

Se seleccionó lanzar la aplicación en otra ventana y esta se abre y se cierra sin que se llegue a ver ninguna otra.	Ha ocurrido algún error y se no se ha podido ejecutar la aplicación. Láncela en la misma ventana y vea entonces cuál es el error que se ha producido para poder subsanarlo.
Al hacer la Monitorización de estados se han quedado todos los componentes en rojo y la ventana de información no para de dar el error A.4.1.3.2.	Posiblemente el agente no responda porque haya tenido una excepción o se haya quedado *colgado*. Vuelva a lanzarlo.
Al realizar una petición de una	El SNMP Tree se lanzó sin hacer

variable SNMP aparece una ventana de confirmación que indica que no hay dirección IP donde mandar la PDU.	Monitorización de estados, y no posee ninguna dirección IP. Lance la Monitorización de estados y cuando el componente a gestionar pase a verde, repita la petición.
Al realizar una petición de una variable SNMP no aparece nada.	El SNMP Tree se lanzó sin tener ningún componente seleccionado. Seleccione alguno que esté en verde (debe haber sido lanzada la Monitorización de estados) y repita la operación.

A.5. Referencias

[Advent98] Advent Network Management, Inc., *Advent Agent Builder Documentation*, http://www.adventnet.com/products/agentbuilder/help/index.html, 1998.

[Iona96] Iona Technologies, Inc., *OrbixWeb Programming Guide*, noviembre de 1996.

[Macaray96] J. F. Macaray y C. Nicolas, *Programación Java*, Gestión 2000, Barcelona, junio de 1996.

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, revisión 2.0, julio de 1995

[Schönwälder95] J. Schönwälder, H. Langendörfer. *Tcl Extensions for Network Management Applications*, Third Tcl/Tk WorkShop, Toronto, Canada, julio de 1995

[SunSoft94] SunSoft, *MIB for the SunOS SNMP Agent included with SunNet Manager 2.2.3*, 1994.

[Welch95] B. Welch, *Practical Programming in Tcl and Tk*, Prentice Hall, California, 1995.

Apéndice B. Listados

B.1. ABSMIBv2-MIB

```
ABSMIBv2-MIB DEFINITIONS ::= BEGIN

IMPORTS TimeTicks             FROM RFC1155-SMI
               DisplayString  FROM RFC1213-MIB;

org            OBJECT IDENTIFIER ::= { iso 3 }
dod            OBJECT IDENTIFIER ::= { org 6 }
internet       OBJECT IDENTIFIER ::= { dod 1 }
private OBJECT IDENTIFIER ::= { internet 4}
mgmt           OBJECT IDENTIFIER ::= { internet  2}
enterprises    OBJECT IDENTIFIER ::= { private 1}

mib-2          OBJECT IDENTIFIER
--  AGENTCLAUSE
--      "
--              PROXY-COMMAND:
--              AGENT-HOST: localhost
--              AGENT-PORT: 30000"
--      END AGENTCLAUSE
        ::= { mgmt 1 }

sun            OBJECT IDENTIFIER
--  AGENTCLAUSE
--      "
--              PROXY-COMMAND:
--              AGENT-HOST: localhost
--              AGENT-PORT: 30000"
--      END AGENTCLAUSE
        ::= { enterprises 42 }

upm            OBJECT IDENTIFIER ::= {enterprises 2781}
abs            OBJECT IDENTIFIER ::= {upm 2}

configuration  OBJECT IDENTIFIER ::= {abs 1}
aa             OBJECT IDENTIFIER ::= {abs 2}
bam            OBJECT IDENTIFIER ::= {abs 3}
bsa            OBJECT IDENTIFIER ::= {abs 4}
bsm            OBJECT IDENTIFIER ::= {abs 5}
cnm            OBJECT IDENTIFIER ::= {abs 6}
cpsa           OBJECT IDENTIFIER ::= {abs 7}
fm             OBJECT IDENTIFIER ::= {abs 8}
msa            OBJECT IDENTIFIER ::= {abs 9}
om             OBJECT IDENTIFIER ::= {abs 10}
pm             OBJECT IDENTIFIER ::= {abs 11}
qee            OBJECT IDENTIFIER ::= {abs 12}
qpg            OBJECT IDENTIFIER ::= {abs 13}
qr             OBJECT IDENTIFIER ::= {abs 14}
```

```
rm              OBJECT IDENTIFIER ::= {abs 15}
usa             OBJECT IDENTIFIER ::= {abs 16}

-- configuration group
absConfTable   OBJECT-TYPE
        SYNTAX  SEQUENCE OF AbsConfTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Information about localisation of Broker System Components"
        ::={configuration 1}

absConfTableEntry     OBJECT-TYPE
        SYNTAX  AbsConfTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Sequence of information about ABS broker application"
        INDEX   {absConfIndex}
        ::={absConfTable 1}

AbsConfTableEntry ::=
        SEQUENCE {
                absConfIndex   INTEGER,
                absConfName    DisplayString,
                absConfHost    DisplayString,
                absConfStatus  INTEGER
        }
absConfIndex   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Index of AbsConfTable"
--    AGENTCLAUSE
--                "
--                CLASS-COMMAND:
--                CLASS-NAME:  ABS.BrokerSystem.ServiceSession.BSM.Configuration
--                INSTANTIATE: at-start"
--   END AGENTCLAUSE
        ::= {absConfTableEntry 1}

absConfName    OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the components whose information is being
                retrieved"
-- AGENTCLAUSE
--              "
```

```
--              CLASS-COMMAND:
--              CLASS-NAME:  ABS.BrokerSystem.ServiceSession.BSM.Configuration
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {absConfTableEntry 2}

absConfHost     OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the host in which each component is running"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.Configuration
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {absConfTableEntry 3}

absConfStatus  OBJECT-TYPE
        SYNTAX  INTEGER {
                        up(1),
                        down(2),
                        unmanageable(3)}
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "indication of whether the component is running or not"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.Configuration
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {absConfTableEntry 4}

-- aa group
aaTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
```

```
        ::=     {aa 1}

-- bam group
bamTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bam 1}

-- bsa group
bsaTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bsa 1}

-- bsm group
bsmConfTable   OBJECT-TYPE
        SYNTAX  SEQUENCE OF BsmConfTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Information about localisation of Broker System Components"
        ::={bsm 1}

bsmConfTableEntry     OBJECT-TYPE
        SYNTAX  BsmConfTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Sequence of information about ABS broker application"
        INDEX   {bsmConfIndex}
        ::={bsmConfTable 1}
```

```
BsmConfTableEntry ::=
        SEQUENCE {
                bsmConfIndex   INTEGER,
                bsmConfName    DisplayString,
                bsmConfHost    DisplayString,
                bsmConfStatus  INTEGER
        }

bsmConfIndex   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Index of BsmConfTable"
--    AGENTCLAUSE
--                "
--                CLASS-COMMAND:
--                CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--                INSTANTIATE: at-start"
--   END AGENTCLAUSE
        ::= {bsmConfTableEntry 1}

bsmConfName    OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the components whose information is being
                retrieved"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmConfTableEntry 2}

bsmConfHost    OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the host in which each component is running"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmConfTableEntry 3}
```

```
bsmConfStatus  OBJECT-TYPE
        SYNTAX  INTEGER {
                        up(1),
                        down(2),
                        unmanageable(3)}
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "indication of whether the component is running or not"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmConfTableEntry 4}

bsmTimeBetweenDaemonCalls     OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time between calling Orbix daemon to obtain the activeServers"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bsm 2}

bsmTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bsm 3}

bsmEventForwarderDiscriminator       OBJECT-TYPE
        SYNTAX  INTEGER { false (0),
                        true (1) }
        ACCESS  read-write
```

```
        STATUS  mandatory
        DESCRIPTION
                "The BSM will act as an EFD if this variable is true"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bsm 4}

bsmLogFileName OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "The name of the file where to log the notifications"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {bsm 5}

bsmNotificationTable   OBJECT-TYPE
        SYNTAX  SEQUENCE OF BsmNotificationTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Information about every notifications in the last session"
        ::={bsm 6}

bsmNotificationTableEntry     OBJECT-TYPE
        SYNTAX  BsmNotificationTableEntry
        ACCESS  not-accessible
        STATUS  mandatory
        DESCRIPTION
                "Sequence of information about any notification"
        INDEX   {bsmNotificationIndex}
        ::={bsmNotificationTable 1}

BsmNotificationTableEntry ::=
        SEQUENCE {
                bsmNotificationIndex   INTEGER,
                bsmNotificationTime          TimeTicks,
                bsmNotificationType    DisplayString,
                bsmNotificationName    DisplayString,
                bsmNotificationArguments      DisplayString
        }
```

```
bsmNotificationIndex   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Index of BsmNotificationTable"
--    AGENTCLAUSE
--                "
--                CLASS-COMMAND:
--                CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--                INSTANTIATE: at-start"
--   END AGENTCLAUSE
        ::= {bsmNotificationTableEntry 1}

bsmNotificationTime    OBJECT-TYPE
        SYNTAX  TimeTicks
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Time from the launch when the notification was received"
--    AGENTCLAUSE
--                "
--                CLASS-COMMAND:
--                CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--                INSTANTIATE: at-start"
--   END AGENTCLAUSE
        ::= {bsmNotificationTableEntry 2}

bsmNotificationType    OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the notification received"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmNotificationTableEntry 3}

bsmNotificationName    OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Name of the components who send the notification"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
```

```
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmNotificationTableEntry 4}

bsmNotificationArguments      OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Arguments of the notification"
-- AGENTCLAUSE
--              "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcherTable
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {bsmNotificationTableEntry 5}

-- cnm group
cnmNumNavigationRequests      OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of Navigation Requests processed by
                        the NE since it last started up"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {cnm 1}

cnmNumNavigationFailures      OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of Navigation Requests that couldn't
                        be processed by the NE since it last started up"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {cnm 2}

cnmDepthLevel  OBJECT-TYPE
```

```
                SYNTAX  INTEGER
                ACCESS  read-write
                STATUS  mandatory
                DESCRIPTION
                        "Maximun time to make a navigation request"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {cnm 3}


cnmTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {cnm 4}

-- cpsa group
cpsaTimeBetweenNotifications  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {cpsa 1}


-- fm group
fmTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
```

```
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {fm 1}


-- msa group
msaTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {msa 1}


-- om group
omMostUsedCP   OBJECT-TYPE
        SYNTAX  DisplayString
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "The most used CP from the point of view of the OM"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {om 1}

omNumofConnections    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of connections at a time"
-- AGENTCLAUSE
--      "
```

```
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {om 2}

omTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {om 3}

-- pm group
pmNumCurrentProfiles   OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of User profiles currently maintained
                        by the PM"
        ::= {pm 1}

pmNumProfilesActivated OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of User profiles activated since
                        the PM last started up"
        ::= {pm 2}

pmNumProfilesDeactivated OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of User profiles deactivated since
                        the PM last started up"
        ::= {pm 3}

pmTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
```

```
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {pm 4}

-- qee group
qeeAvPlanTime  OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Average time consumed in the execution of a Plan"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {qee 1}

qeeAvSubGoalTime       OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Average time consumed in the execution of a SubGoal"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::= {qee 2}

qeeTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
```

```
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qee 3}

-- qpg group
qpgMostfreqViewpoint   OBJECT-TYPE
                SYNTAX  DisplayString
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Most frequently asked viewpoint"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 1}

qpgMostfreqSO  OBJECT-TYPE
                SYNTAX  DisplayString
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Most frequently asked SO"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 2}

qpgNumofreceivedQuery  OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS  read-only
                STATUS  mandatory
                DESCRIPTION
                        "Number of Received Queries"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 3}

qpgNumofUnresolvedqueries     OBJECT-TYPE
        SYNTAX  INTEGER
```

```
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of unresolved queries due to the lack of
                information"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 4}

qpgNumofPlans  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of the generated plans"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 5}

qpgPlanGenerationTime  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Plan generation processing time"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 6}


qpgTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
```

```
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qpg 7}

-- qr group
qrTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {qr 1}

-- rm group
rmNumofNewOffers      OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of new offers"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {rm 1}
rmNumofModifiedOffers  OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of modified offers"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {rm 2}
rmNumofWithdrawOffers  OBJECT-TYPE
        SYNTAX  INTEGER
```

```
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of deleted offers"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {rm 3}
rmNumofRegisteredCP   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-only
        STATUS  mandatory
        DESCRIPTION
                "Number of registered CPs in the RM"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {rm 4}

rmTimeBetweenNotifications    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {rm 5}


-- usa group
usaAvQueryTime OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS read-only
                STATUS  mandatory
                DESCRIPTION
                        "Average duration of a query"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
```

```
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {usa 1}

usaAvNavigTime OBJECT-TYPE
                SYNTAX  INTEGER
                ACCESS read-only
                STATUS  mandatory
                DESCRIPTION
                        "Average duration of a navigation operation"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {usa 2}

usaTimeBetweenTraps    OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {usa 3}

usaTimeBetweenNotifications   OBJECT-TYPE
        SYNTAX  INTEGER
        ACCESS  read-write
        STATUS  mandatory
        DESCRIPTION
                "Time in seconds between sending trap number 4 (componentIsReady)
                time=0 means that it will just be sent when launch"
-- AGENTCLAUSE
--      "
--              CLASS-COMMAND:
--              CLASS-NAME: ABS.BrokerSystem.ServiceSession.BSM.AbsDispatcher
--              INSTANTIATE: at-start"
-- END AGENTCLAUSE
        ::=     {usa 4}


-- Traps proposed
```

```
unknownManagementVariable     TRAP-TYPE
                              ENTERPRISE abs
                              VARIABLES {absConfName}
                              DESCRIPTION
                                      "An unknownManagementVariable trap means that
                                      the variable asked for does not exist"
                              ::= 1

genericSoftwareFailure TRAP-TYPE
                       ENTERPRISE abs
                       VARIABLES {absConfName}
                       DESCRIPTION
                              "A genericSoftwareFailure trap means that
                              an element of the system has thrown an exception"
                              ::= 2


setVariableNotAllowed  TRAP-TYPE
                       ENTERPRISE abs
                       VARIABLES {absConfName}
                       DESCRIPTION
                              "A setVariableNotAllowed trap means that
                              the variable tried to set is read-only"
                              ::= 3

componentIsReady       TRAP-TYPE
                       ENTERPRISE abs
                       VARIABLES {
                              absConfName,
                              absConfHost,
                              absConfStatus
                       }
                       DESCRIPTION
                              "A componentIsReady trap means that
                              an element of the system is now running"
                              ::= 4

END
```

B.2. BSMMgmtHandler.java

```
// BSMMgmtHandler.java
// Handler to manage the BSM.
// 30-6-98
// jlopez@dit.upm.es

package ABS.BrokerSystem.MgmtSession.MGMT;

import ABS.CORBASystem.ABSMIBv2_MIB.*;
import ABS.CORBASystem.*;

import org.omg.CORBA.Any;
import org.omg.CORBA.ORB;
import org.omg.CORBA.DATA_CONVERSION;
import org.omg.CORBA.SystemException;

public class BSMMgmtHandler
        extends SmiEntryImplementation
        implements _bsmOperations {

        //Management variables
        public int bsmTimeBetweenDaemonCalls;
        public int bsmTimeBetweenNotifications;
        public boolean bsmEventForwarderDiscriminator;
        public String bsmLogFileName;

        //Management tables
        public BSMConfTable bsmConfTable;
        public BSMNotificationTable bsmNotificationTable;

        bsm tie=null;


        public BSMMgmtHandler () {
                // Register to the orbix daemon
                try {
                        tie= new _tie_bsm(this);
                } catch (SystemException se) {
                        System.out.println("BSM: Unable to create new tie");
                        System.out.println(se.toString());
                        System.exit(1);
                }

                // Register to NameService
                try {
                        bind("bsm","0",tie); // inherited method from SmiEntry
                } catch (Exception e) {
```

```
                        System.out.println("BSM:   Unable   to   bind   the   service   in   the
NameService");
                        System.out.println("BSM: BSM won't be manageable");
                }

                // Check the table contexts, to unbind them if necessary
                checkContext("bsmConfTableEntry");
                checkContext("bsmNotificationTableEntry");
        }

        public int bsmTimeBetweenDaemonCalls(){
                return bsmTimeBetweenDaemonCalls;
        }

        public void bsmTimeBetweenDaemonCalls(int value){
                bsmTimeBetweenDaemonCalls=value;
        }

        public int bsmTimeBetweenNotifications(){
                return bsmTimeBetweenNotifications;
        }

        public void bsmTimeBetweenNotifications(int value){
                bsmTimeBetweenNotifications=value;
        }

        public int bsmEventForwarderDiscriminator(){
                if (bsmEventForwarderDiscriminator) return 1;
                else return 0;
        }

        public void bsmEventForwarderDiscriminator(int value){
                if (value==0) bsmEventForwarderDiscriminator=false;
                else bsmEventForwarderDiscriminator=true;
        }

        public byte[] bsmLogFileName(){
                return bsmLogFileName.getBytes();
        }

        public void bsmLogFileName(byte[] value){
                bsmLogFileName=new String(value);
                System.out.println("bsmLogFileName ->"+bsmLogFileName);
        }

// Redefine define_property and get_property
        public void define_property(String property_name,
                org.omg.CORBA.Any property_value)
                throws  ABS.CORBASystem.CosPropertyService.InvalidPropertyName,
                        ABS.CORBASystem.CosPropertyService.ConflictingProperty,
                        ABS.CORBASystem.CosPropertyService.UnsupportedTypeCode,
```

```
                ABS.CORBASystem.CosPropertyService.UnsupportedProperty,
                ABS.CORBASystem.CosPropertyService.ReadOnlyProperty,
                org.omg.CORBA.SystemException {

        if (property_name.equals("bsmTimeBetweenDaemonCalls")) {
                try {
                        bsmTimeBetweenDaemonCalls=property_value.extract_long();
                } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error extracting value");
                }
        } else if (property_name.equals("bsmTimeBetweenNotifications")) {
                try {
                        bsmTimeBetweenNotifications=property_value.extract_long();
                } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error extracting value");
                }
        } else if (property_name.equals("bsmEventForwarderDiscriminator")) {
                try {
                        int tmp=property_value.extract_long();
                        if (tmp==0) bsmEventForwarderDiscriminator=false;
                        else bsmEventForwarderDiscriminator=true;
                } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error extracting value");
                }
        } else if (property_name.equals("bsmLogFileName")) {
                try {
                        bsmLogFileName=new String(
                                ASN1_OctetStringHelper.extract(property_value));
                } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error extracting value");
                }
        } else
                throw new ABS.CORBASystem.CosPropertyService.InvalidPropertyName();
}

public org.omg.CORBA.Any get_property_value(String property_name)
        throws  ABS.CORBASystem.CosPropertyService.PropertyNotFound,
                ABS.CORBASystem.CosPropertyService.InvalidPropertyName,
                org.omg.CORBA.SystemException {

        org.omg.CORBA.Any result=ORB.init().create_any ();

        if (property_name.equals("bsmTimeBetweenDaemonCalls")) {
                try {
                        result.insert_long(bsmTimeBetweenDaemonCalls);
                        return result;
                } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error inserting value");
                }
        } else if (property_name.equals("bsmTimeBetweenNotifications")) {
                try {
```

```
                        result.insert_long(bsmTimeBetweenNotifications);
                        return result;
                    } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error inserting value");
                    }
                } else if (property_name.equals("bsmEventForwarderDiscriminator")) {
                    try {
                        if (bsmEventForwarderDiscriminator) result.insert_long(1);
                        else result.insert_long(0);
                        return result;
                    } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error extracting value");
                    }
                } else if (property_name.equals("bsmLogFileName")) {
                    try {
                        ASN1_OctetStringHelper.insert(result,bsmLogFileName.getBytes());
                        return result;
                    } catch (SystemException se) {
                        throw new DATA_CONVERSION("Error inserting value");
                    }
                } else
                    throw new ABS.CORBASystem.CosPropertyService.InvalidPropertyName();
        }
}
```

Pliego de condiciones

Normas de obligado cumplimiento

El presente proyecto se ajusta a las especificaciones relacionadas con el modelo de gestión de red de Internet, así como la arquitectura CORBA y las Traducciones SNMP-CORBA propuestas por JIDM. Los documentos que los definen son los que siguen:

[Rose90] M. T. Rose and K. McCloghrie. *Structure and Identification of Management Information for TCP/IP based internets*. Request For Comments 1155, Performance Systems International, Inc., 1990.

[Case90] J. D. Case, M. S. Fedor, M. L. Schoffstall y J. R. Davin, *A Simple Network Management Protocol*. Request For Comments 1157, SNMP Research, Inc., 1990.

[Rose91] M. Rose, K. McCloghrie, *Concise MIB Definitions*, Request for Comments 1212, Performance Systems International, marzo de 1991.

[McCloghrie91] K. McCloghrie and M. T. Rose. *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, Request for Comments 1213, Hughes LAN Systems, Inc., 1991.

[Rose91] M. T. Rose, *A convention for Defining Traps for use with the SNMP*, RFC 1215, Performance Systems International, marzo de 1991.

[OMG95] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, revisión 2.0, julio de 1995

[Open97] The Open Group, *Inter-domain Management: Specification Translation*. Open Group Preliminary Specification P509, marzo de 1997.

[OMG98] Object Management Group, *CORBA/TMN Interworking Final Submission – JIDM-SNMP*, OMG document telecom/98-05-03, mayo de 1998.

Otras condiciones

En la fase de desarrollo de este proyecto se ha pretendido gestionar la Aplicación de Intermediación del consorcio ABS. Por tanto, los modelos de información de gestión empleados se han ajustado a las especificaciones que presenta la arquitectura de ABS.

Otras condiciones aplicables son:

- La transferencia de la información de gestión debe hacerse siguiendo el protocolo de gestión SNMP y el protocolo IIOP de interoperabilidad entre objetos de CORBA.
- Se simulará el sistema mediante objetos CORBA que posean la funcionalidad de gestión que define su grupo de la MIB.

Para el desarrollo del sistema de gestión se necesita la disponibilidad previa de los siguientes paquetes de *software*:

- *Advent Networks Inc. SNMP Extensible Agent & SNMP Stack*: Agente Extensible y pila SNMP desarrollados en Java.
- *SunNet Manager Agent*: Agente de red y sistema para estaciones de trabajo *Sun*.
- *Scotty*: Plataforma de gestión de red desarrollada en TCL/TK.
- *OrbixWeb 3.0*: Implementación de la plataforma de procesamiento distribuido CORBA.

Medios Materiales

El proyecto ha sido llevado a cabo utilizando el sistema operativo Solaris, de *Sun*. El *Software* que constituye el desarrollo del Sistema de Gestión se ha realizado usando los lenguajes de programación Java y TCL.

Para el desarrollo de la totalidad del proyecto se ha utilizado el siguiente conjunto de herramientas:

- Compilador e intérprete del paquete *JDK1.1.5* para Java
- Intérprete para TCL.
- Editor de textos *dtpad*.
- Utilidades asociadas al comando *make*.
- *smi2idl*: Traductor de MIBs según los algoritmos descritos por JIDM-ST.
- *PC-Xware95, v3*: Servidor de Ventanas X para Windows 95
- *Rational Rose98 Enterprise Edition*: Herramienta CASE para desarrollo de aplicaciones orientadas a objetos.

Así mismo se ha contado con el siguiente conjunto de equipos.

- Estación de Trabajo Sun Sparc Ultra 1 con 64 MBytes de RAM y sistema operativo Solaris 2.5.1. actuando como máquina servidora.
- Ordenador 486 PC-Compatible, con 16 MBytes de RAM y sistema operativo Windows 95 actuando como terminal.

Presupuesto

1. Presentación del proyecto

El presente proyecto describe el diseño e implementación de un sistema de gestión basado en SNMP para la aplicación de intermediación electrónica basada en CORBA que se está implementando en el ámbito del proyecto europeo ACTS - ABS (*Architecture for Information Brokerage Service*, Arquitectura para un Servicio de Intermediación de Información).

El sistema implementado trata de monitorizar y controlar los recursos empleados en la provisión del servicio de intermediación ABS. Este Sistema de Gestión se ha diseñado con la idea de soportar las cinco áreas funcionales incluidas el modelo FCAPS del modelo OSI-SM de gestión de red: fallos, configuración, contabilidad, rendimiento y seguridad.

Además, adopta un paradigma de integración basado en la unificación de la red, el sistema, la aplicación y aspectos de servicios de gestión. En otras palabras, en un mismo sistema de gestión se puede controlar todos los aspectos de gestión. Para poder conseguir esa integración, y con la meta de reutilizar tantos componentes existentes como sea posible, se han elegido los estándares de gestión de Internet (los llamados SNMP, *Simple Network Management Protocol*) como base para el sistema de gestión.

Para ello se basa en el uso de una pasarela SNMP-CORBA que usa los algoritmos de traducción especificados por JIDM (*Joint Inter-Domain Management*, Unión para la Gestión Inter-Dominios) e incorpora ideas de los servicios y funcionalidades contenidas en la parte de SNMP de los documentos de Traducción de interacción.

Este proyecto también enfoca cómo se ha instrumentado la información de gestión en la aplicación CORBA de forma tan trasparente como ha sido posible respecto a los desarrolladores de los aspectos funcionales de la aplicación gestionada.

2. Descomposición en tareas y plazos

Tarea 1 (T1)

- **Objetivo**: Estudio del estado del arte sobre gestión de sistemas distribuidos.
- **Duración**: 2 meses.
- **Esfuerzo**: Ingeniero superior: 1 h-m.
- **Resultado**: Adquisición de los conocimientos teóricos necesarios para abordar el proyecto.
- **Comienzo**: A la firma del contrato.

Tarea 2 (T2)

- **Objetivo**: Familiarización y estudio de herramientas y entorno.
- **Duración**: 1 mes.
- **Esfuerzo**: Ingeniero superior: 0,25 h-m. Ingeniero Técnico: 0,75 h-m.
- **Resultado**: Adquisición de la práctica para poder afrontar las fases de desarrollo.
- **Comienzo**: Dado que no es necesario esperar a que acabe la tarea T1, puesto que ya se conoce la máquina y el software a utilizar, se comienza al mes del comienzo de la tarea T1.

Tarea 3 (T3)

- **Objetivo**: Análisis del Sistema de Gestión.
- **Duración**: 1 mes.
- **Esfuerzo**: Ingeniero Superior: 1 h-m.
- **Resultado**: Captura de los requisitos del sistema.
- **Comienzo**: Al acabar T1.

Tarea 4 (T4)

- **Objetivo**: Diseño del Sistema de Gestión.
- **Duración**: 2 meses.
- **Esfuerzo**: Ingeniero Superior: 1 h-m. Ingeniero Técnico: 0,25 h-m.
- **Resultado**: Estructura del Sistema de gestión.
- **Comienzo**: Tras acabar T3.

Tarea 5 (T5)

- **Objetivo**: Implementación del Sistema de Gestión.
- **Duración**: 2 meses.
- **Esfuerzo**: Ingeniero Superior: 0,1 h-m. Ingeniero Técnico: 1 h-m.
- **Resultado**: Versión preliminar del Sistema de Gestión.
- **Comienzo**: Al acabar T4.

Tarea 6 (T6)

- **Objetivo**: Pruebas
- **Duración**: 2 meses.
- **Esfuerzo**: Ingeniero Superior: 0,1 h-m. Ingeniero Técnico: 1 h-m.
- **Resultado**: Versión final del Sistema de Gestión.
- **Comienzo**: Dado que se puede comenzar al ir acabando cada módulo, el comienzo se produce 1 mes después del comienzo de T5.

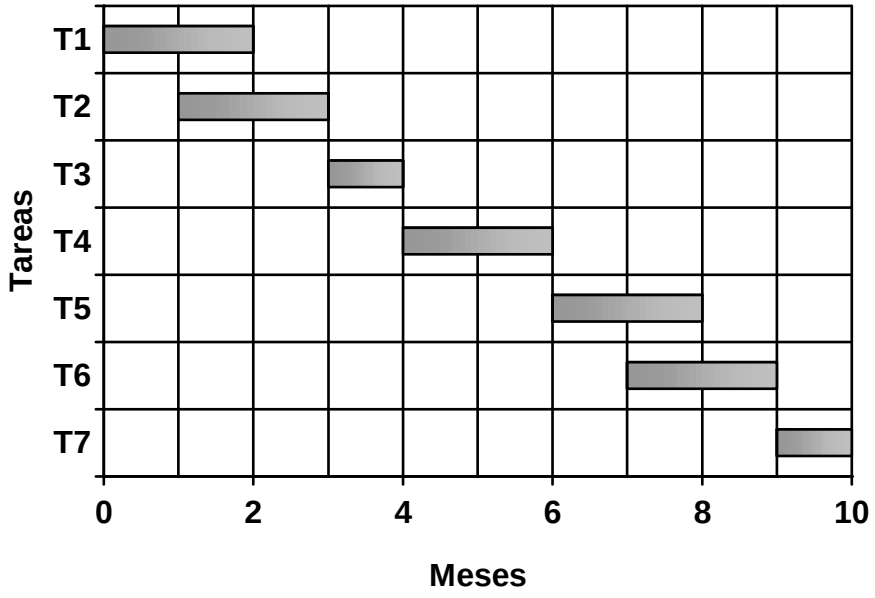
Tarea 7 (T7)

- **Objetivo**: Documentación del proyecto

- **Duración**: 1 mes.
- **Esfuerzo**: Ingeniero Superior: 0,5 hombres/mes. Administrativo: 0,5 hombres/mes.
- **Resultado**: Obtención del presente documento.
- **Comienzo**: Al finalizar T6.

Diagrama de Gantt de las tareas

Tras definir las tareas, y realizar el diagrama Gantt de dependencias se obtiene que el proyecto tiene una duración de 10 meses.



3. Costes

Costes de personal

Los costes de los distintos perfiles profesionales necesarios para la ejecución del proyecto por hora de trabajo son:

- Ingeniero superior (5000 Pts/hora) (29,76 €/hora[1])
- Ingeniero técnico (3000 Pts/hora) (17,86 €/hora)
- Administrativo (1000 Pts/hora) (5,95 €/hora)

Basándose en esta división del trabajo, en la siguiente tabla se presenta la asignación de tiempos por tareas para cada uno de los miembros del equipo, así como el coste de las mismas:

Tareas	Ingeniero Superior (h-m)[2]	Ingeniero Técnico (h-m)	Administrativo (h-m)	Coste	
				Pesetas	Euros
T1	1	0	0	1008000	6000,00
T2	0,25	0,75	0	630000	3750,00
T3	1	0	0	1008000	6000,00
T4	1	0,25	0	1134000	6750,00
T5	0,1	1	0	604800	3600,00
T6	0,1	1	0	604800	3600,00
T7	0,5	0	0,5	588000	3500,00
Totales				5577600	33200,00

[1] Se supone el cambio 1 Euro=168 Pesetas. Este valor es meramente informativo

[2] Duración en horas sobre una jornada de 8 horas/día y 21 días/mes

El coste de mano de obra se eleva a la cantidad de CINCO MILLONES QUINIENTAS SETENTA Y SIETE MIL SEISCIENTAS pesetas.

Costes de material

Los costes materiales del proyecto debido al equipo utilizado se consideran en la siguiente tabla:

Concepto	Coste	
	Pesetas	Euros
Ordenador PC de gama alta (*Hardware y Software*)	300000	1785,71
Estación de Trabajo Sun Ultra-1 (*Hardware y Software*)	1500000	8928,57
Licencia de *OrbixWeb 3.0 Professional*	224850	1338,39
Licencia de *Rational Rose 98 Enterprise Edition*	540000	3214,29
Licencia de software de *Advent Network Inc.* (Agente Extensible + Pila SNMP)	375000	2232,14
Material Fungible	100000	595,24
Total	3039850	18094,35

Los costes de material ascienden a TRES MILLONES TREINTA Y NUEVE MIL OCHOCIENTAS CINCUENTA pesetas.

Presupuesto Total

Por último, los costes totales resultantes, incluyendo el IVA, son:

	Coste	
	Pesetas	Euros
Costes de Personal	5577600	33200,00
Costes Materiales	3039850	18094,35
Total	8617450	51294,35
IVA (16%)	1378792	8207,10
Coste Final Total	9996242	59501,44

El presupuesto total de este proyecto asciende a NUEVE MILLONES NOVECIENTAS NOVENTA Y SEIS MIL DOSCIENTAS CUARENTA Y DOS pesetas.

Madrid, 9 de septiembre de 1998.

El ingeniero proyectista

Fdo: Jorge Enrique López de Vergara Méndez