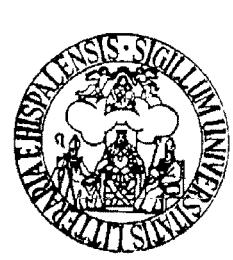
TESIS DOCTORAL



CONTRIBUCIONES A LA GESTIÓN DE DISPOSITIVOS EN SISTEMAS BASADOS EN CORBA

por

Teresa Ariza Gómez

Licenciada en Informática por la Facultad de Informática y Estadística

de la Universidad de Sevilla

Presentada en la

Escuela Superior de Ingenieros

de la

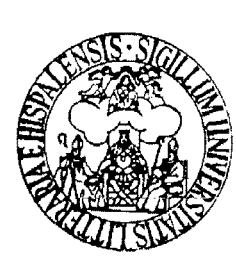
Universidad de Sevilla

para la obtención del

Grado de Doctor Informático

Sevilla, 2000

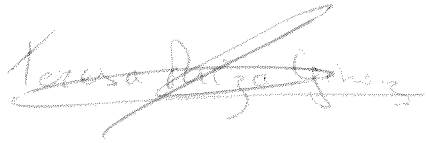
TESIS DOCTORAL



CONTRIBUCIONES A LA GESTIÓN DE DISPOSITIVOS EN SISTEMAS BASADOS EN CORBA

Autor: Teresa Ariza Gómez

Director: Francisco Rodríguez Rubio



FDO. TERESA ARIZA GÓMEZ

Fdo. Francisco Rodríguez Rubio

Resumen

El desarrollo de dos tecnologías fundamentales para el desarrollo de aplicaciones como son los sistemas distribuidos y el paradigma orientado a objetos ha dado lugar a un nuevo paradigma: los objetos distribuidos. Los objetos distribuidos son cápsulas de información que pueden realizar servicios requeridos de forma remota por otros objetos. El Modelo de Referencia de Procesamiento Distribuido y Abierto (RM-ODP) propone en su modelo de ingeniería una plataforma para la comunicación entre objetos distribuidos de forma transparente. La Arquitectura Común del Agente de Peticiones de Objetos (CORBA) se considera una aproximación a este modelo realizada por el Grupo de Gestión de Objetos (OMG).

Debido a la difusión que ha tenido CORBA en los últimos años y el gran número de implementaciones que han surgido, tanto en la industria como en la comunidad de investigadores se ha elegido esta arquitectura como base para facilitar la implementación de aquellas aplicaciones que son inherentemente distribuidas.

Por otro lado, se está estudiando la posibilidad de utilizar CORBA en plataformas o arquitecturas que tradicionalmente utilizan otras formas de comunicación. Este es el caso de la gestión de sistemas. Se han realizado diversos estudios relacionados con la aplicación de los sistemas de objetos distribuidos tanto a la gestión de red como a la gestión de sistemas. De forma que se pueden integrar componentes CORBA en sistemas de gestión tradicionales y también integrar en plataformas de gestión basadas en CORBA elementos que trabajan con protocolos tradicionales de gestión.

De esta forma se consiguen sistemas de gestión más flexibles donde es más fácil integrar todos los componentes del sistema distribuido, no sólo los elementos propiamente de la red sino todos aquellos que intervienen como parte de la aplicación distribuida.

El objetivo de esta tesis es estudiar la integración de dispositivos de fabricación en sistemas de gestión basados en CORBA. En concreto se estudia la posibilidad de utilizar el protocolo estándar de Especificación de Mensajes de Fabricación (MMS) utilizado en sistemas de manufactura para la comunicación entre los distintos componentes como base para esta integración.

Para alcanzar el objetivo de esta tesis, previamente se realiza un estudio del estado del arte de todas las tecnologías implicadas. Se realiza un estudio del estado del arte de la arquitectura CORBA, de los sistemas de gestión y del protocolo MMS.

Una vez que se conocen las posibilidades de cada una de las tecnologías utilizadas, se expone un modelo de integración de los dispositivos de fabricación en los sistemas de gestión que usan CORBA como base para la comunicación.

También se trata de la adaptación de MMS a CORBA que se necesita realizar para la utilización de este protocolo como base para el desarrollo de la interfaz de los objetos distribuidos de fabricación.

Se pone especial atención en la comunicación de eventos. Para ello se proponen dos formas distintas de difundir estos eventos. La primera es mediante la interfaz proporcionada por MMS para el modelo de eventos y una segunda es utilizar mecanismos propios de CORBA como el servicio de eventos.

Por último, se presenta también el prototipo implementado durante el desarrollo de esta tesis para dar un carácter práctico a los estudios teóricos que se han realizado.

Agradecimientos

Quisiera hacer constar mi agradecimiento al Departamento de Ingeniería de Sistemas y Automática por proporcionarme el tiempo y los medios necesarios para desarrollar este trabajo. En especial quiero agradecer a mi director de Tesis, D. Francisco Rodríguez Rubio, sus constantes consejos y comentarios que han servido para el desarrollo de esta tesis, así como el continuo (y paciente) seguimiento del trabajo realizado.

También quisiera expresar mi especial reconocimiento a D. Juan Manuel Vozmediano Torres por "regalarme" el libro en el que descubrí las maravillas de los Objetos Distribuidos, a D. Juan Antonio Ternero Muñiz que ha sido mi compañero en las aventuras y desventuras del comienzo del Área de Ingeniería Telemática y a D. José Manuel Fornés Rumbao por su inestimable ayuda.

Agradezco también a todos los compañeros del Área su apoyo incondicional y ánimos prestados que tanto me ayudaron a pasar el periodo de tiempo en que estuve dedicada a la Tesis.

Por último en esta lista, aunque los primeros en mi corazón, quisiera incluir a mi familia y amigos, en particular a mi marido, que me ha dado el apoyo y el equilibrio necesarios para poder dedicar mi pensamiento a esta difícil tarea.

Glosario

CIM	Modelo de Información Común. *Common Information Model.*
COD	Código Bajo Demanda. *Code On Demand.*
CORBA	Arquitectura Común del Agente de Peticiones de Objetos. *Common Object Request Broker Architecture.*
COSS	Servicios Comunes de Objetos. *Common Object Services.*
CMIP	Protocolo de Información de Gestión Común. *Common Management Information Protocol.*
CMIS	Servicio de Información de Gestión Común. *Common Management Information Service.*
DAI	Inteligencia Artificial Distribuida. *Distributed Artificial Intelligence.*
DCE	Entorno de Computación Distribuida. *Distributed Computing Environment.*
DCOM	Modelo Distribuido de Objetos Componentes. *Distributed Component Object Model.*
DII	Interfaz de Invocación Dinámica. *Dynamic Invocation Interface.*
DN	Nombre Distinguido. *Distinguished Name.*
DMTF	Grupo de Trabajo de Gestión Distribuida. *Distributed Management Task Force.*
DPE	Entorno de Procesamiento Distribuido. *Distributed Processing Environment.*
ESIOP	Protocolos Inter-ORB Específicos del Entorno. *Environment-Specific Inter-ORB Protocols.*
GDMO	Guía para la Definición de Objetos Gestionados. *Guidelines for the Definition of Managed Objects.*
GIOP	Protocolo Inter-ORB General. *General Inter-ORB Protocol.*

IDL	Lenguaje de Definición de Interfaz. *Interface Definition Language.*
IETF	Grupo de Trabajo en Ingeniería Internet. *Internet Engineering Task Force.*
IIOP	Protocolo Inter-ORB Internet. *Internet Inter-ORB Protocol.*
IOR	Referencia a Objeto Interoperable. *Interoperable Object Reference.*
ISO	Organización Internacional de Normalización. *International Standards Organization.*
ISO/IEC	ISO/Comisión Electrotécnica Internacional. *ISO/International Electrotechnical Commision.*
ITU	Unión Internacional de Telecomunicaciones. *International Telecommunication Union.*
ITU-T	ITU, Sector de Telecomunicación. *ITU Telecommunication Sector.*
JCP	Proceso de la Comunidad Java. *Java Community Process.*
JIDM	Unión de Gestión Inter-Dominio. *Joint Inter-Domain Management.*
JMAPI	API de Gestión Java. *Java Management API.*
JMX	Extensiones Java para Gestión. *Java Management Extensions.*
KQML	Lenguaje de Manipulación y Petición de Conocimiento. *Knowledge Query and Manipulation Language.*
LRPC	Llamada a Procedimiento Remoto Ligera. *Lightweight Remote Procedure Call.*
MA	Agentes Móviles. *Mobile Agent.*
MBeans	Componentes Gestionados. *Managed Beans.*
MF	Bloque Funcional de Mediación. *Mediation Functions.*
MIB	Base de Información de Gestión. *Management Information Base.*
MMS	Especificación de Mensajes de Fabricación. *Manufacturing Message Specification.*
NCCE	Entorno de Comunicaciones y Computación Nativa. *Native Computing and Communications Environment.*
NEF	Bloque Funcional de Elemento de Red. *Network Element Functions.*
NMF	Foro de Gestión de Red. *Network Management Forum.*

OCX	Controles Clientes. *Custom Controls.*
ODL	Lenguaje de Descripción de Objetos. *Object Description Language.*
ODP	Procesamiento Distribuido y Abierto. *Open Distributed Processing.*
OLE	Objetos Enlazados e Integrados. *Object Linking and Embedding.*
OMA	Arquitectura de Gestión de Objetos. *Object Management Architecture.*
OMG	Grupo de Gestión Abierta. *Open Management Group.*
ONA	Arquitectura de Red Abierta. *Open Network Architecture.*
OO	Orientado a Objetos. *Object Oriented.*
ORB	Agente de Peticiones de Objetos. *Object Request Broker.*
OSF	Bloque Funcional de Sistema de Operación. *Operations System Functions.*
PDU	Unidad de Datos del Protocolo. *Protocol Data Unit.*
POA	Adaptador de Objetos Portable. *Portable Object Adapter.*
QAF	Bloque Funcional de Adaptador Q. *Q Adaptor Functions.*
REV	Evaluación Remota. *Remote EValuation.*
RDN	Nombre Distinguido Relativo. *Relative Distinguished Name.*
RFC	Petición de Comentarios. *Request For Comments.*
RMI	Invocación de Métodos Remotos. *Remote Method Invocation.*
RM-ODP	Modelo de Referencia - ODP. *Reference Model - ODP.*
RPC	Llamada a Procedimiento Remoto. *Remote Procedure Call.*
SII	Interfaz de Invocación Estática. *Static Invocation Interface.*
SNMP	Protocolo de Gestión de Red Simple. *Simple Network Management Protocol.*
SMI	Estructura de Información de Gestión. *Structure of Management Information.*

TC	Comité Técnico. *Technical Committee.*
TCP/IP	Protocolo de Control de Transmisión/Protocolo Internet. *Transmission Control Protocol/Internet Protocol.*
TINA	Arquitectura de Red de la Información de Telecomunicaciones. *Telecommunications Information Networking Architecture.*
TMN	Red de Gestión de Telecomunicaciones. *Telecommunications Management Network.*
WBEM	Gestión Basada en Web. *Web-Based Management.*
WMI	Instrumentación de Gestión Windows. *Windows Management Instrumentation.*
WSF	Bloque Funcional de Estación de Trabajo. *Work Station Functions.*

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este capítulo se comenta el marco general de la tesis introduciendo las tecnologías bases que se han utilizado, así como la motivación y los objetivos de ésta. En el último apartado se detalla la estructura completa de la tesis.

1.1 Marco general

Actualmente, los sistemas de computación distribuida están siendo usados en una gran variedad de entornos, debido a los grandes avances logrados en las redes de ordenadores en los últimos años, y en parte debido al bajo coste de éstas.

Un sistema distribuido [Tan95, Cor91] está compuesto por diversos elementos (ordenadores y otros dispositivos) que trabajan juntos para lograr realizar una tarea común. Entre las ventajas de un sistema distribuido frente a un sistema centralizado se pueden mencionar las siguientes:

- *Económicas*: Es más barato conectar varios ordenadores que disponer de un único ordenador potente con la misma potencia de cálculo que la suma de los anteriores.
- *Velocidad*: Todos los componentes de un sistema distribuido trabajando juntos pueden lograr un mejor rendimiento.
- *Distribución inherente*: Hay aplicaciones que son inherentemente distribuidas, como por ejemplo las aplicaciones de fabricación.

- *Fiabilidad*: Los sistemas centralizados dependen de un único ordenador central, mientras que un sistema distribuido es más tolerante a fallos, puesto que si un componente falla, no significa que el sistema completo se pare.

- *Crecimiento Incremental*: Cuando sea necesario, el sistema puede crecer añadiendo potencia de computación, sin tener que desechar el sistema anterior sino mediante la adición de nuevos elementos.

Además de estas ventajas, igual que en los sistemas centralizados, tanto los datos como los dispositivos pueden ser compartidos y los usuarios se pueden comunicar de una forma fácil, pero ganando en flexibilidad.

A pesar de estas ventajas, deben ser mencionas algunas desventajas:

- *Heterogeneidad*: Los componentes que componen un sistema distribuido son normalmente de diferentes fabricantes, y sus funciones son diferentes.

- *Comunicación*: En los sistemas distribuidos se debe tratar con la complejidad adicional de la interconexión de los distintos componentes que conlleva la adición de distintos niveles de comunicación.

- *Software*: El desarrollo de software para estos sistemas llega a ser complicado, principalmente debido a la heterogeneidad del sistema distribuido y a los niveles de comunicación subyacente.

Otro avance tecnológico que ha contribuido a un cambio importante en los sistemas de computación es el paradigma orientado a objetos (OO) [Inc91]. Este paradigma ofrece una visión de la aplicación como una agregación de objetos. Un objeto es una encapsulación abstracta de información junto con los métodos o procedimientos para manipularla. Los objetos con las mismas características se agrupan para formar una clase que es una generalización de un tipo específico de objetos. Este paradigma proporciona las siguientes ventajas:

- *Reutilización del software*: En los proyectos tradicionales, es difícil extraer piezas de código para ser utilizadas en otros proyectos, debido a que las diferentes partes del código son interdependientes. Sin embargo, utilizando metodologías orientas a objetos, los objetos son implementados y depurados de forma independiente, de forma que pueden ser integrados en cualquier proyecto que necesite la misma funcionalidad proporcionada por estos objetos.

- *Integración*: La integración de las diferentes partes de un proyecto es más directa. Debido a que los componentes que deben ser integrados son los objetos y que estos presentan una interfaz bien definida, la integración se puede realizar de una forma más fácil.

- *Facilidad en la depuración y el mantenimiento*: Esto se consigue debido a la división de la aplicación en objetos y la creación de éstos de forma independiente.

Estas dos tecnologías, los sistemas distribuidos y la orientación a objetos, se unen para dar paso a los objetos distribuidos que cuenta con las ventajas de ambos. En este marco surge el Modelo de Referencia para el Procesamiento Distribuido y Abierto (RM-ODP) de ISO. CORBA [Orf96] es considerado una aproximación al modelo de ingeniería de este modelo de referencia y está siendo utilizado para facilitar la implementación de aplicaciones distribuidas [Ari99a].

MMS [Pim90, Val92] es también considerado como parte de estas tecnologías. Es un protocolo del nivel de aplicación para sistemas distribuidos heterogéneos, que homogeneiza el uso de dispositivos que componen un sistema de fabricación. MMS hace uso de la aproximación de objetos para especificar los servicios que un usuario puede invocar para comunicar con estos dispositivos.

Por otra parte, los sistemas de gestión han cobrado importancia en la última década, ya que está probada su utilidad en la configuración de sistemas distribuidos, así como en la monitorización y en la corrección y detección de fallos. Debido a la influencia que tienen los sistemas de objetos distribuidos en la gestión de sistemas, ésta ha ido evolucionando desde los protocolos tradicionales (SNMP y CMIP principalmente) a otros esquemas más flexibles soportados por estas arquitecturas de comunicación más modernas.

1.2 Motivación y objetivos

Como se ha comentado anteriormente, los sistemas de gestión están evolucionando hacia esquemas más flexibles, esto permite integrar en la gestión otros dispositivos que no sean exclusivamente disposivos de red y considerar además el conjunto de dispositivos que participan en las aplicaciones distribuidas. La motivación principal de este trabajo es el estudio de las aplicaciones distribuidas de fabricación desde el punto de vista de la gestión de estos sistemas.

Más concretamente, el objetivo de esta tesis es presentar un modelo para la integración de dispositivos de fabricación en sistemas de gestión basados en CORBA

utilizando la aproximación MMS para estructurar el sistema en objetos y presentar una interfaz a estos objetos, centrando el trabajo principalmente en la comunicación de eventos. Se pretende con ello:

- Conseguir ampliar la gestión para que abarque dispositivos de fabricación además de dispositivos de red.
- Aprovechar las facilidades que proporciona CORBA para la implementación de sistemas distribuidos.
- Utilizar el protocolo MMS como base para la definición de objetos y sus interfaces.
- Integrar los dispositivos de fabricación en los nuevos sistemas emergentes para la gestión de sistemas que utilizan CORBA como arquitectura que permite la comunicación entre los distintos componentes del sistema.
- Estudiar las distintas posibilidades que se pueden plantear para la comunicación de eventos generados en los distintos dispositivos a la aplicación de gestión.

Por otra parte, para lograr los objetivos se debe realizar un estudio detallado de la arquitectura CORBA, de la gestión de sistemas y la influencia del paradigma de objetos distribuidos en ésta, así como del protocolo MMS y sus posibles aportaciones para la gestión.

1.3 Estructura de la Tesis

Esta tesis consta de 8 capítulos y 3 apéndices. Este primer capítulo es una introducción al resto de los capítulos donde se ha dado una vista general de los motivos de la tesis.

En el segundo, tercer y cuarto capítulos se trata de las tecnologías base y se expone el estado del arte de estas tecnologías.

En el segundo capítulo se trata de CORBA, introduciendo el Modelo de Referencia de Procesamiento Distribuido y Abierto (RM-ODP), detallando los componentes de la arquitectura de Gestión de Objetos (OMA) y exponiendo la estructura del Agente de Peticiones de Objetos (ORB) que es el componente más importante de esta arquitectura. Se comenta la evolución de CORBA y se mencionan otras plataformas distribuidas que sirven también para la comunicación de procesos en sistemas distribuidos pero que no han cobrado tanta importancia como ésta.

En el tercer capítulo se explican las bases de los sistemas de gestión, su importancia, los protocolos que tradicionalmente se han utilizado y la influencia de CORBA en estos sistemas, así como otras tendencias modernas que están siendo estudiadas en la actualidad.

En el cuarto capítulo se introduce el protocolo MMS, detallando las distintas clases de objetos que se incluyen en el estándar. Este capítulo da una idea de la utilidad de este protocolo para el desarrollo de aplicaciones distribuidas en entornos de fabricación.

En el quinto capítulo se expone el modelo de integración propuesto para conseguir la gestión de dispositivos de fabricación, el modelo de objetos y de información para la integración, así como los componentes arquitecturales del modelo. Se comentan otras posibles alternativas para la integración que aunque quedan fuera del ámbito de esta tesis, podrían ser retomadas en trabajos futuros.

En el sexto se detalla la adaptación del protocolo MMS a CORBA, se presenta la adaptación del modelo de objetos, del modelo de comunicación y de los servicios de MMS. También se presenta el esquema de traducción de las estructuras de datos de MMS a IDL.

En el séptimo se exponen los modelos de la comunicación de eventos generados por los dispositivos de fabricación en un entorno CORBA. Se tratan dos modelos diferentes, uno basado en la traducción de los servicios MMS y otro basado en la utilización del servicio de eventos de CORBA para la transmisión de estos eventos.

Para finalizar, el último capítulo contiene las contribuciones y las conclusiones. Se ha incluido también posibles vías de investigación futura que tienen como base el trabajo realizado en esta tesis.

Debido a que tanto IDL como ASN.1 han sido muy utilizados en el desarrollo de esta tesis, se han incluido en el apéndice A y en el B respectivamente para aclarar las explicaciones realizadas en los demás capítulos. En el apéndice C se explica el prototipo que ha sido desarrollado como base de esta tesis, este prototipo ha sido implementado en Java y permite aplicar los modelos que se han propuesto en este trabajo.

Capítulo 2

Estado del Arte de CORBA

En este capítulo se va a justificar la importancia del avance de las plataformas de objetos distribuidos, introduciendo el modelo de referencia ODP y presentando las principales características de la Arquitectura Común del Agente de Peticiones de Objetos (CORBA, Common Object Request Broker Architecture), ya que esta arquitectura ha sido tomada como base para el desarrollo de esta tesis. Finalmente se resumen las principales características de otras arquitecturas distribuidas, justificando la elección de CORBA como soporte para la comunicación.

2.1 Plataformas de Objetos Distribuidos

Las plataformas de objetos distribuidos tienen su base conceptual en un paradigma bien asentado en los últimos años, la orientación a objetos, que presenta excelentes cualidades técnicas como la separación explícita entre interfaces e implementaciones y la posibilidad de generalización de implementaciones para fomentar la reutilización. Sobre este paradigma se ha definido un modelo de referencia estándar para el procesamiento distribuido y abierto (RM-ODP, Reference Model- Open Distributed Processing) que goza de consenso internacional y ha sido asumido por otros modelos de arquitecturas de referencia para sistemas de telecomunicación (por ejemplo, TINA).

Finalmente, también se ha producido consenso a nivel industrial para la disponibilidad de implementaciones de la plataforma distribuida que sirve de base al modelo ODP, como es el caso del estándar industrial OMG-CORBA.

Los Objetos Distribuidos son cápsulas de información que pueden vivir en cualquier lugar de la red. Los clientes pueden acceder a ellos a través de invocación de métodos,

siendo transparente el sistema operativo, el compilador y la plataforma hardware utilizada.

Las ventajas que se han podido apreciar en la utilización de plataformas de objetos distribuidos son las siguientes:

- Es la tecnología más adecuada para crear sistemas cliente/servidor flexibles.
- Los objetos son entidades autogestionadas.
- Posibilita la interoperatividad a través de la red.
- Los objetos pueden funcionar en distintas plataformas.

Por todo ello y tras un exhaustivo análisis del estado del arte, las plataformas de objetos distribuidos se han seleccionado como la mejor tecnología de partida posible para abordar este trabajo de tesis.

2.2 RM-ODP

El modelo de referencia ODP fue un esfuerzo conjunto de ISO e ITU-T para desarrollar un marco de coordinación para la normalización del procesamiento distribuido y abierto. Crea una arquitectura dentro de la cual se puede integrar un soporte de distribución, cooperación, interoperabilidad y portabilidad.

Los avances en redes de ordenadores han permitido que sistemas de computación repartidos por todo el mundo hayan sido interconectados. Pero a pesar de esto, la heterogeneidad en los modelos de interacción impiden la cooperación entre sistemas. ODP describe sistemas que soportan procesamiento distribuido heterogéneo dentro de y entre organizaciones mediante el uso de un modelo de interacción común.

El ámbito de interés para ODP es, en resumen, la provisión de un marco conceptual y de soporte para la construcción de sistemas distribuidos y abiertos a partir de un conjunto de sistemas en red que son heterogéneos por naturaleza. La heterogeneidad mencionada puede ser de varios tipos: de equipo, de sistema operativo, de lenguaje, de aplicación y de propiedad de los recursos.

2.2.1 Objetivos

El modelo de referencia para el procesamiento distribuido y abierto pretende lograr:

- Portabilidad de aplicaciones sobre plataformas heterogéneas.
- Cooperación entre sistemas ODP, es decir, correcto intercambio de información y uso adecuado de la funcionalidad a través de sistemas distribuidos.
- Transparencia de distribución, es decir, ocultar las consecuencias de la distribución a ambos, a los programadores de aplicaciones y a los usuarios.

2.2.2 Estructura del Estándar

El modelo de referencia de procesamiento distribuido abierto está constituido por:

- Rec. ITU-T X.901 — ISO/IEC 10746-1 [ISO94a]: Visión de conjunto: contiene una visión de conjunto de las motivaciones de ODP, que da el alcance, la justificación y la explicación de conceptos esenciales, y una descripción de la arquitectura ODP. Contiene material explicativo sobre la interpretación y aplicación de RM-ODP por los usuarios, que pueden incluir escritores de normas y arquitectos de sistemas ODP. Contiene también una agrupación en categorías de las áreas requeridas de normalización, expresadas en términos de los puntos de referencia para conformidad identificados en la Rec. acerca de la arquitectura. Esta parte no es normativa.
- Rec. ITU-T X.902 — ISO/IEC 10746-2 [ISO94b]: Fundamentos: contiene la definición de los conceptos y marco analítico para la descripción normalizada de sistemas de procesamiento distribuido. Introduce los principios de conformidad con las normas ODP y la forma en que deben aplicarse. La exposición se hace solamente a un nivel de detalle suficiente para la aplicación de la Rec. acerca de la arquitectura y el establecimiento de los requisitos que cumplirán las nuevas técnicas de especificación. Esta parte es normativa.
- Rec. ITU-T X.903 — ISO/IEC 10746-3 [ISO94c]: Arquitectura: contiene la especificación de las características que debe tener un procesamiento distribuido para que sea abierto. Emplea las técnicas descriptivas de la Rec. acerca de los fundamentos. Esta parte es normativa.
- Rec. UIT-T X.904 — ISO/IEC 10746-4 [ISO94d]: Semántica arquitectural: contiene una formalización de los conceptos de modelado ODP definidos en esta

Recomendación — Norma Internacional. La formalización se consigue interpretando cada concepto en término de las construcciones de las diferentes técnicas de descripción formal normalizadas. Esta parte es normativa.

2.2.3 Puntos de Vista

La tercera parte del modelo de referencia para procesamiento distribuido y abierto describe el uso de puntos de vista para abstraer los sistemas ODP. Para cada uno de los puntos de vista se da un conjunto de conceptos, estructuras y reglas, proporcionando un lenguaje para la especificación de sistemas ODP para cada una de estas perspectivas. RM-ODP define cinco puntos de vista diferentes que se van a comentar brevemente a continuación:

- *Modelo de empresa*

  Se usa para recoger las necesidades de los usuarios finales del sistema de información. En la especificación de empresa de una aplicación ODP se determinan las políticas de uso y los requisitos de empresa. No se realizan imposiciones debido a las elecciones de tecnología. Las políticas son definidas en términos de:

  - Objetos tanto activos como pasivos.
  - Comunidades, que son agrupaciones de objetos con el fin de lograr algún propósito.
  - Roles de los objetos en las comunidades, expresadas en términos de políticas de permisos, prohibiciones y obligaciones.

- *Modelo de información*

  Se usa para describir la información requerida por una aplicación ODP. El estado y la estructura de un objeto se describe mediante el uso de esquemas:

  - Esquema estático: captura el estado y la estructura de un objeto en algún instante en particular.
  - Esquema invariante: especifica restricciones que debe cumplir siempre el estado y la estructura de un objeto.
  - Esquema dinámico: define un cambio permitido en el estado y la estructura de un objeto. Un esquema dinámico está siempre restringido por los esquemas invariantes.

  Los esquemas se pueden también usar para describir relaciones y asociaciones entre objetos. Se pueden describir objetos complejos o compuestos mediante la composición de un conjunto de esquemas.

La especificación de la información podría ser expresada usando una gran variedad de métodos, por ejemplo, utilizando el modelo entidad-relación, lenguajes de especificación formal o técnicas de modelado de objetos.

- *Modelo computacional*

  Este modelo es usado para especificar la funcionalidad de la aplicación de forma transparente a la distribución. Esta perspectiva está basada en objetos, y se compone de los siguientes elementos:

  - Objeto computacional: Encapsulación de datos y procesamiento (comportamiento).
  - Interfaz computacional: Es ofrecida por un objeto para permitir la interacción con otros objetos.
  - Objeto de asociación: Soporta una vinculación entre una conjunto de objetos computacionales.

  La especificación computacional define los objetos que se encuentran en el sistema, las actividades en estos objetos, y las interacciones entre ellos. La mayoría de los objetos describen funcionalidad de la aplicación. Estos objetos son conectados mediante enlaces a través de los cuales se producen las interacciones. Los objetos de enlace se usan para describir interacciones complejas entre objetos.

  Los objetos computacionales modelan tanto los componentes de aplicación que realizan procesamiento de información como aquellos otros que almacenan dicha información.

- *Modelo de ingeniería*

  Una especificación de ingeniería define los mecanismos y funciones requeridos para soportar la interacción distribuida entre objetos en un sistema ODP. No trata acerca de la semántica de la aplicación, excepto para determinar sus requerimientos para la distribución y la transparencia de distribución. Proporciona un entorno de ejecución para aplicaciones distribuidas independiente de las máquinas y la tecnología utilizadas.

  El conjunto de servicios básicos y mecanismos identificados en la perspectiva de ingeniería se modelan como un conjunto de *objetos de ingeniería* que proporcionan el soporte necesario para posibilitar la interacción entre componentes de una aplicación distribuida. Se identifican distintos tipos de objetos de ingeniería que se corresponden con distintas funciones de distribución. Los objetos y estructuras definidos en el modelo de ingeniería de RM-ODP son los siguientes:

  - Objeto de ingeniería básico: Corresponden a objetos de la especificación computacional.
  - Conglomerado: Conjunto de objetos de ingeniería básicos que forman una sola unidad para fines de desactivación, reactivación, recuperación y traslado.

- Cápsula: Conjunto de objetos de ingeniería que forman una sola unidad para fines de encapsulación, procesamiento y almacenamiento. Un ejemplo de cápsula es un proceso.
- Núcleo: Objeto de ingeniería que coordina funciones de procesamiento, almacenamiento y comunicaciones para uso por otros objetos de ingeniería dentro del nodo al que pertenece. Un ejemplo de núcleo es un sistema operativo.
- Nodo: Conjunto de objetos de ingeniería que forman una sola unidad a los fines de ubicación en el espacio, y que incorpora un conjunto de funciones de procesamiento, almacenamiento y comunicación. Un ejemplo de nodo es un computador.
- Canal: Conjunto de objetos que proporcionan una asociación entre un conjunto de interfaces de objetos de ingeniería. Un canal se compone de los siguientes elementos:
  - Stub: Objeto de ingeniería que interpreta las interacciones transportadas por el canal y realiza cualquier transformación o supervisión necesaria. Actúa de representante de un objeto de ingeniería básico, frente a otro objeto localizado en un conglomerado diferente, contribuyendo de esta forma a la transparencia de distribución.
  - Vinculador: Objeto de ingeniería que mantiene una vinculación distribuida entre objetos de ingeniería básicos que interactúan.
  - Interceptor: Objeto de ingeniería que supervisa las políticas sobre interacciones permitidas entre objetos de ingeniería en dominios diferentes y efectúa las transformaciones para enmascarar diferencias en la interpretación de datos por objetos de ingeniería básicos en dominios diferentes.
  - Objeto de protocolo: Objeto de ingeniería que comunica con otros objetos de protocolo en el mismo canal para lograr la interacción entre objetos de ingeniería básicos.

- *Modelo de tecnología*

  En este modelo se describe la implementación del sistema y la información necesaria para la prueba.

2.2.4 Transparencias de Distribución ODP

La transparencia de distribución es un requisito importante en los sistemas distribuidos, ésta se refiere a la propiedad de ocultar a un determinado usuario el comportamiento potencial de algunas partes de un sistema distribuido. RM-ODP define un conjunto de transparencias de distribución que hacen posible la implementación de sistemas ODP

que son transparentes a la distribución desde el punto de vista de los usuarios de esos sistemas. Las siguientes transparencias son contempladas en este modelo de referencia:

- Transparencia de acceso: Esta transparencia posibilita la interoperabilidad en arquitecturas y lenguajes de programación heterogéneos. Enmascara las diferencias en la representación de datos y en los mecanismos de invocación para permitir el interfuncionamiento entre objetos.
- Transparencia de fallo: Con esta transparencia se oculta el fallo y la posible recuperación de otros objetos (o la suya propia), contribuyendo de esta manera a soportar tolerancia a fallos.
- Transparencia de ubicación: Enmascara la utilización de información sobre la ubicación en el espacio cuando se efectúa la identificación y la vinculación a interfaces. Permite a los objetos tener acceso a las interfaces sin utilizar información de ubicación.
- Transparencia de traslado: Esta transparencia enmascara, para que no sea percibida por un objeto, la aptitud de un sistema para cambiar la ubicación de ese objeto. El traslado puede utilizarse para conseguir el equilibrio de la carga y reducir los efectos de errores no detectados.
- Transparencia de persistencia: Se utiliza para que no sea percibida por un objeto la desactivación y reactivación de otros objetos (o las suyas propias). La desactivación y reactivación a menudo se utilizan para asegurar la persistencia de un objeto cuando un sistema es incapaz de proporcionarla continuamente con funciones de procesamiento, almacenamiento y comunicación.
- Transparencia de reubicación: Con este mecanismo se oculta al usuario los efectos de la migración de un objeto a otra localización entre invocaciones consecutivas del mismo servicio.
- Transparencia de replicación: Oculta la utilización de un grupo de objetos mutuamente compatibles en comportamiento, para soportar una interfaz. Es decir, coordina un conjunto de réplicas de forma que aparezca ante los objetos con los que interactúa como si fuera una sola. La replicación se utiliza a menudo para mejorar el rendimiento y la disponibilidad.
- Transparencia de transacción: Enmascara la coordinación de las actividades de los objetos que constituyen una configuración de objetos, para lograr la consistencia.

Las transparencias definidas en RM-ODP son punto de comienzo para los requerimientos comunes, no pueden ser entendidas como el conjunto completo. Transparencias adicionales, tanto para necesidades generales y específicas podrían ser estandarizadas en el futuro.

2.3 Arquitectura de Gestión de Objetos

La Arquitectura de Gestión de Objetos (OMA, Object Management Architecture) [Sol95] ha sido propuesta por el Grupo de Gestión Abierta (OMG, Open Management Group) que está compuesto por más de 800 vendedores, desarrolladores y usuarios de software. La misión de este grupo es promover la teoría y la práctica de la tecnología de objetos para el desarrollo de sistemas de computación distribuida. El objetivo es proporcionar un marco para una arquitectura común para aplicaciones orientadas a objetos basada en especificaciones de interfaces disponibles. Desde 1989, el Grupo de Gestión de Objetos ha estado trabajando para crear estándares de componentes software basados en objetos en el marco de trabajo de su Arquitectura de Gestión de Objetos. El componente clave es la Arquitectura Común del Agente de Peticiones de Objetos (CORBA 1.1) [Obj95b], cuya especificación fue adoptada en 1991. En 1994, en CORBA 2.0 se definió la interoperabilidad entre objetos en sistemas heterogéneos.

La arquitectura propuesta por el Grupo de Gestión de Objetos es ilustrada en la figura 2.1 y sus componentes se comentan brevemente a continuación:

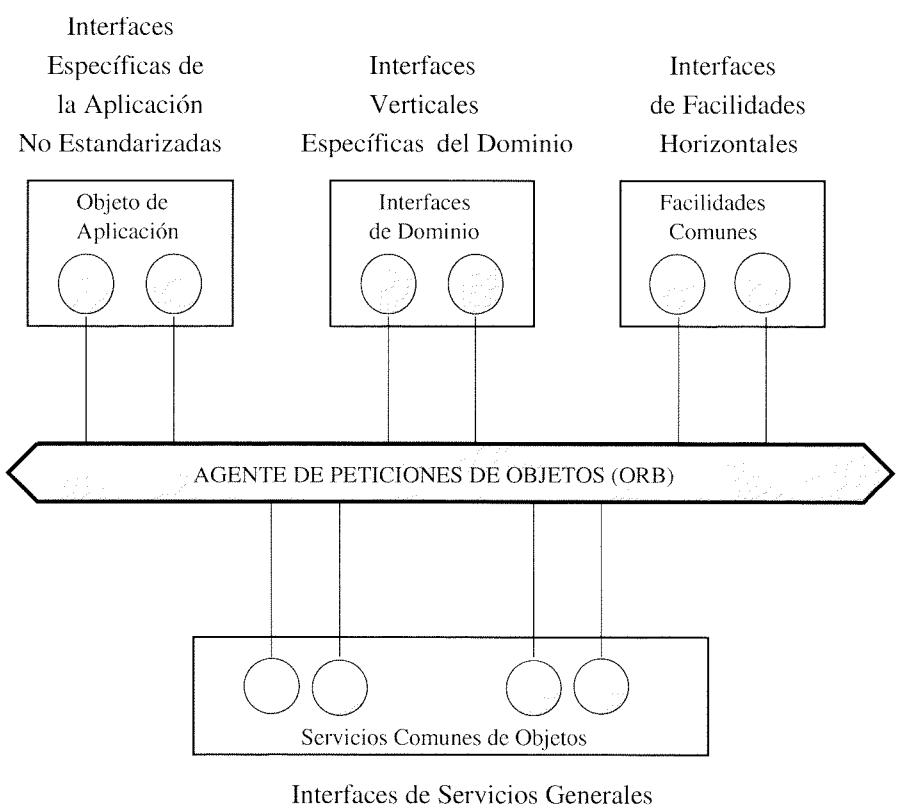


Figura 2.1: Arquitectura de Gestión de Objetos.

Agente de Peticiones de Objetos (ORB)

También conocido como el Bus de Objetos, proporciona el mecanismo para que los objetos realicen peticiones y obtengan respuestas a estas peticiones de una forma transparente. El agente de peticiones de objetos (ORB, Object Request Broker) simplifica la programación de objetos distribuidos ya que libera al cliente de implementar los mecanismos necesarios para las invocaciones de métodos remotos del servidor. Este elemento consigue que, desde el punto de vista del cliente, una llamada a un método remoto se realice de forma similar a la llamada a un método local.

El ORB no es un único elemento, sino que se compone de varias interfaces. Cualquier implementación del ORB debe ajustarse a estas interfaces.

Servicios Comunes de Objetos (COSS)

Los Servicios Comunes [Obj95c] son objetos que ofrecen servicios fundamentales y de uso frecuente en diversas aplicaciones. Se encargan de aumentar y complementar la funcionalidad del ORB. Son considerados servicios de bajo nivel (en contraste a las Facilidades Comunes).

Hasta ahora el estándar ha definido los siguientes:

- Ciclo de vida: Define las operaciones para crear, copiar, mover y borrar componentes del bus.
- Persistencia: Almacena componentes permanentemente en una variedad de servidores almacén como por ejemplo una base de datos.
- Servicio de nombrado: Permite a unos componentes localizar a otros por su nombre. Esto lleva a crear directorios o contextos de nombrado.
- Servicio de eventos: Permite a los componentes demostrar dinámicamente su interés por participar o no en eventos específicos. Este servicio posee un objeto, el canal de eventos, que recoge las peticiones de estos eventos y las distribuye entre los componentes interesados.
- Servicio de control de concurrencia: Controla el tráfico y puede producir un paro en algún sentido, para favorecer una cierta transacción.
- Servicio de relaciones: Crea asociaciones dinámicas entre componentes que no se conocen, y también mecanismos para moverse por las uniones creadas que forman grupos a través de esas asociaciones.

- Servicio de externalización: Provee un modo de introducir u obtener datos de componentes, usando un pseudo-mecanismo que imita al real, es decir, como un flujo de datos.

- Servicio de información: Para atender las preguntas o requerimientos de los objetos.

- Servicio de licencia: Lleva control del uso de los componentes para asegurar una compensación adecuada. Impone recargos por instanciación de ciertos objetos, por sesión de trabajo con un objeto, etc.

- Servicio de propiedades: Dinámicamente se pueden asociar propiedades a un estado de un componente como, por ejemplo, el título.

Facilidades Comunes

Las Facilidades Comunes [Obj95a] son objetos que ofrecen servicios (facilidades) de más alto nivel para el uso directo de los objetos de aplicación (pueden usar o especializar varios módulos COSS). Se configuran de acuerdo a la aplicación y se sitúan más cerca del usuario (entorno de trabajo). Son independientes del dominio de aplicación (interfaz de usuario, impresión, bases de datos, documentos compuestos, etc).

Interfaces de Dominios

Representan áreas verticales que proporcionan funcionalidades de interés para usuarios finales en dominios de aplicaciones particulares. Son dependientes del dominio de la aplicación (telecomunicaciones, fabricación, finanzas, etc).

Objetos de Aplicación

Los Objetos de Aplicación son componentes específicos de las aplicaciones de usuarios finales. Estos objetos deben ser definidos usando IDL para poder participar en los intercambios mediados por el ORB. Una aplicación surge como resultado del ensamblado de estos objetos. Los Objetos de Aplicación están construidos encima de los servicios proporcionados por el ORB, de las Facilidades Comunes y de los Servicios Comunes.

2.4 Lenguaje de Definición de Interfaz

Dentro de un mismo programa, lenguaje, S.O. es fácil que se unan objetos para colaborar, pero cuando el sistema es distribuido la cosa evidentemente se complica. Deben entenderse aún escritos en lenguajes distintos o funcionando en plataformas hardware y software diferentes.

Claramente se advierte la necesidad de estándares, que definan interfaces, para que los objetos interoperen en medios cliente/servidor heterogéneos. Algo loable de OMG es que creó esos estándares antes que los objetos distribuidos se desarrollaran más.

El lenguaje de Definición de Interfaz (IDL, Interface Definition Language) se utiliza para especificar los límites de los componentes y sus interfaces con los clientes. IDL es un lenguaje neutral y totalmente declarativo (no define detalles de implementación). Proporciona interfaces independientes del Sistema operativo y de los lenguajes de programación a todos los servicios y componentes que residen en un bus CORBA.

La definición IDL de un objeto CORBA contiene el conjunto de operaciones que un cliente puede invocar en este objeto, los tipos de datos que se intercambian y las excepciones que pueden ocurrir.

La sintaxis del lenguaje IDL es muy similar a la de lenguajes de programación orientados a objetos como por ejemplo C++ o Java, aunque su simplicidad hace que se pueda definir aproximadamente en 40 páginas. Una definición IDL es independiente del lenguaje en el que se implemente el objeto.

La estructura de la definición IDL de un objeto es la siguiente:

```
Module<identifier> {
   <type declarations>;
   <constant declarations>;
   <exception declarations>;
          interface <identifier> [:inheritance]  {
                     <type declarations>;
                     <constant declarations>;
                     <attribute declarations>;

                     [<op_type>] <identifier>(<parameters>)
                     [raises exception ] [context];
                              :
                     [<op_type>] <identifier>(<parameters>)
                     [raises exception ] [context];
```

```
                                :
        }
interface <identifier> [:inheritance]
                                :
}
```

Un módulo define un contexto de nombrado donde se agrupan un conjunto de declaraciones de tipos, constantes, excepciones e interfaces. Una interfaz define una clase CORBA, donde se indican los métodos contenidos en la interfaz.

Los compiladores de IDL son capaces de mapear una definición IDL a un lenguaje de programación concreto para proporcionar acceso al ORB a los objetos CORBA implementados en ese lenguaje.

Una definición más detallada de IDL se expone en el apéndice A.

2.5 Estructura del ORB

El componente central de la arquitectura de gestión de objetos es el ORB. Su estructura es mostrada en la figura 2.2 y las distintas interfaces y componentes son comentados en los siguientes apartados.

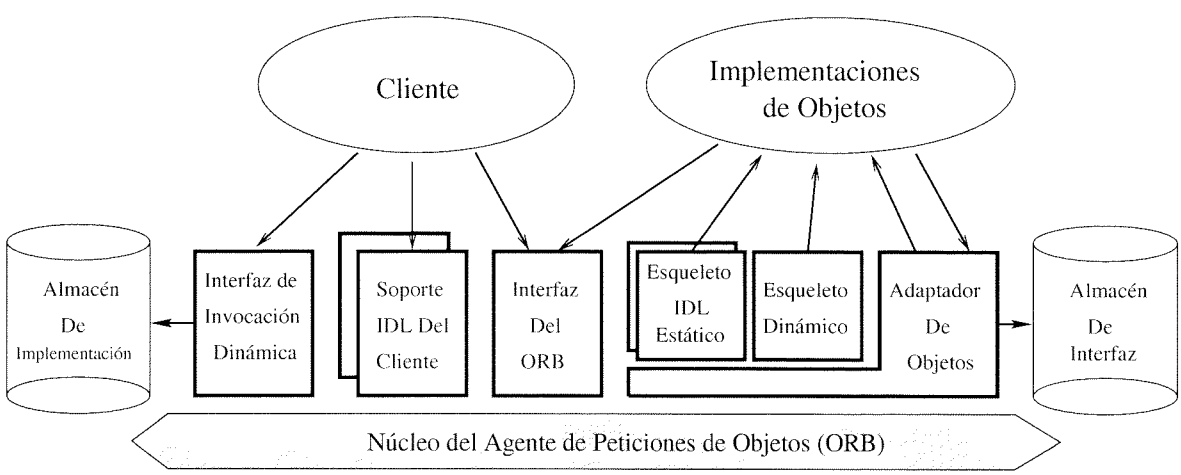


Figura 2.2: Estructura del ORB.

Cliente

Un cliente de un objeto invoca operaciones en el objeto utilizando un referencia a éste. El cliente sólo conoce la estructura lógica del objeto dada por su interfaz.

El cliente puede utilizar dos mecanismos distintos para llamar a los métodos del objeto, la invocación estática (SII, Static Invocation Interface) o la invocación dinámica (DII, Dynamic Invocation Interface).

Los clientes generalmente ven a los objetos y al ORB a través de una perspectiva de mapeado del lenguaje que utilicen, obteniendo una visión del ORB y del objeto en el nivel de programación. El "stub" o soporte del cliente es la interfaz a los servicios del servidor que ve el cliente.

Soporte IDL del Cliente (SII, Static Invocation Interface)

Este componente es utilizado en el lado del cliente cuando se utiliza la invocación estática mediante la interfaz de invocación estática o SII. El soporte se genera a partir de la especificación IDL de la interfaz. Se necesita un compilador de IDL del lenguaje correspondiente (C, C++, Smalltalk, ADA). El código generado se enlaza con el del cliente.

Los tipos de objetos con los cuales se puede interactuar se conocen en tiempo de compilación (la estructura de la petición está pre-construida). Siempre es necesario adquirir la referencia al objeto concreto en tiempo de ejecución. La SII es el caso más frecuente y más simple de invocación. Ofrece la ventaja de que se comprueban los tipos en tiempo de compilación.

Interfaz de Invocación Dinámica(DII, Dynamic Invocation Interface)

Este componente se encuentra también en el lado cliente, pero a diferencia de la invocación estática, cuando se utiliza la invocación dinámica mediante el interfaz de invocación dinámica o DII la petición se construye en tiempo de ejecución con ayuda de los servicios del ORB y del almacén o repositorio de interfaces.

El servidor no puede distinguir si se está usando la SII o la DII. La DII es más flexible que la SII y en algunos casos es inevitable (no se cuenta con la especificación

IDL del servidor, no se quiere enlazar con muchos stubs etc.), pero la programación es más complicada, el uso es mucho menos eficiente y no hay comprobación de tipos de datos.

Esqueleto IDL Estático

Transfiere la llamada hacia el método de la implementación. Básicamente es una interfaz a los métodos que implementa cada tipo de objeto. Es el encargado de realizar la conversión de los parámetros en la llamada y de los valores devueltos en el retorno.

La existencia del esqueleto estático no implica la existencia del correspondiente soporte del cliente, ya que los clientes pueden realizar las peticiones mediante la interfaz de invocación dinámica.

Esqueleto Dinámico

Proporciona un mecanismo para gestionar llamadas a métodos no compilados con IDL. Son útiles en la integración de aplicaciones no basadas en CORBA y para puentes genéricos entre ORBs.

Interfaz del ORB

Es la interfaz a un conjunto de operaciones proporcionadas directamente por el ORB, que son las mismas para todos los ORB's e independientes de la interfaz del objeto y del adaptador de objetos. Estas operaciones son útiles tanto en la parte del cliente como en la parte del servidor.

Almacén de Implementación

Almacena información de administración de la implementación (instalación, localización, activación, permisos ...). Se proporciona un conjunto de comandos interactivos para su manejo y una interfaz IDL para manejarlo desde programa.

Almacén de Interfaz

Almacena información estructurada relativa a las interfaces IDL. Utiliza metadatos CORBA y proporciona el IDL del almacén para que lo puedan utilizar tanto el ORB como los clientes y servidores. El ORB lo puede utilizar para comprobar los datos o signaturas de operaciones, comprobar las relaciones de herencia o para interactuar con otros ORBs. Los clientes o los servidores lo pueden utilizar en la DII para acceder a la información y para modificar su contenido.

Adaptador de Objetos

El adaptador de objetos es el mecanismo principal para que una implementación de objeto pueda acceder a los servicios del ORB. Proporciona un entorno completo para la ejecución de un servidor. Algunos servicios proporcionados por el adaptador de objetos son:

- Registra las clases del servidor en el almacén de implementación. El repositorio de implementación se puede pensar como un almacenamiento persistente que gestiona el adaptador de objetos. Clases de objetos de implementación son registradas y almacenadas en el depósito de la implementación.
- Instancia nuevos objetos en tiempo de ejecución. El adaptador de objetos es responsable de crear instancias de objetos de las clases de implementación. El número de las instancias creadas está en función de la carga del tráfico cliente que llega.
- Genera y gestiona referencias de objetos. El adaptador de objetos asigna referencias (con id único) a los nuevos objetos que crea. Es responsable del mapeado de referencias de objetos específicas de la implementación y específicas del ORB.
- Difunde la presencia de objetos servidores. Es responsable de hacerle saber al mundo exterior los servicios que maneja.
- Maneja llamadas de los clientes. Interactúa con los niveles superiores de la pila de comunicación del núcleo del ORB, pone la petición en una forma entendible por el stub y se la pasa al stub. El stub debe interpretar los parámetros y debe presentarlos en una forma que es aceptable para la invocación del método del objeto.
- Dirige la llamada al método apropiado. El adaptador de objetos está implícitamente involucrado en la invocación de los métodos descritos en los esqueletos. Por ejemplo, el adaptador puede estar involucrado en activar la implementación, y puede autentificar la petición que llega.

Hay una variedad de posibles adaptadores de objetos. Sin embargo, debido a que las implementaciones de los objetos dependen de los adaptadores, es más práctico contar con los imprescindibles, de forma que sólo en el caso de que una implementación requiera un servicio radicalmente diferente, sea considerado el diseñar un nuevo adaptador.

En la especificación CORBA se define el Adaptador de Objetos Portable (POA, Portable Object Adapter). En esta definición, se proporciona un adaptador de objetos que se puede utilizar con diversos ORB's con un mínimo de reescritura necesaria para tratar con las distintas implementaciones.

Se pueden usar distintas políticas para comenzar un servidor:

- Servidor compartido: Se activa el servidor la primera vez que se realiza una petición a un objeto implantado en ese servidor. Después de que el servidor se ha inicializado a sí mismo, notifica al POA que está preparado para recibir peticiones. No se activará otro proceso servidor para esa misma implementación. El servidor sólo maneja una petición cada vez, y notifica al POA cuando ha acabado de gestionarla.

- Servidor no compartido: Cada objeto reside en un proceso de servidor diferente. Se activa un nuevo servidor la primera vez que se realiza una petición al objeto. Un nuevo servidor se inicializa si se hace una petición para un objeto que todavía no está activo, incluso si un servidor para otro objeto de la misma implementación está activo.

- Servidor por método: Se activa un nuevo proceso cada vez que se realiza una nueva petición, haya o no otra petición en curso para esa operación.

2.6 Interoperabilidad

La interoperabilidad del ORB especifica una aproximación flexible para soportar redes de objetos que están distribuidos y gestionados por múltiples y heterogéneos ORB's que cumplen con la especificación CORBA.

Los factores que han motivado la especificación de la interoperabilidad entre ORB's son los siguientes:

- Gran diversidad de técnicas de implementación.

- La partición de un entorno en varios ORB's llega a ser interesante por razones de seguridad, de mantenimiento, y de gestión.
- Los distintos ORB's pueden variar en ámbito, distancia y tiempo de vida, ajustándose a las necesidades de los usuarios.

Los elementos que se especifican para permitir la interoperabilidad son los siguientes:

- Arquitectura de interoperabilidad.
- Soporte para puentes inter-ORB.
- Protocolos inter-ORB General y para Internet (GIOP e IIOP).
- Protocolos inter-ORB Específicos del entorno (ESIOP).

La arquitectura proporciona un marco de trabajo conceptual para la definición de elementos de interoperabilidad y para la identifición de sus puntos comunes. También caracteriza nuevos mecanismos y especifica convenciones necesarias para lograr la interoperabilidad entre ORB's producidos independientemente.

El soporte para puentes inter-ORB especifica las API's y convenciones para permitir la construcción de puentes entre distintos ORB's. Cuando una invocación debe dejar su dominio, la invocación debe atravesar un puente. El papel del puente es asegurar que el contenido y la semántica es mapeada de una forma apropiada de un ORB a otro.

Para pasar referencias a objetos entre ORBs diferentes, se tiene que usar una referencia a objeto especial conocida como Referencia a Objeto Interoperable (IOR, Interoperable Object Reference). Una IOR incluye datos que identifican el dominio del ORB al que esta referencia está asociada.

2.6.1 GIOP

El Protocolo Inter-ORB General (GIOP, General Inter-ORB Protocol) es un elemento que especifica una sintaxis de transferencia y un conjunto de formatos de mensajes para la comunicación entre ORB's. GIOP está específicamente construido para la interacción entre ORB's, y está diseñado para trabajar directamente sobre cualquier protocolo de transporte orientado a conexión que cumpla unos requerimientos mínimos. No se requiere ningún mecanismo de más alto nivel, como por ejemplo las RPC's. El protocolo es simple, escalable y relativamente fácil de implementar.

2.6.2 IIOP

El Protocolo Inter-ORB Internet (IIOP, Internet Inter-ORB Protocol) es un elemento que especifica como los mensajes GIOP son intercambiados usando una conexión TCP/IP. El protocolo es diseñado para que sea apropiado para ser usado por cualquier ORB que opere en los dominios del protocolo Internet.

2.6.3 ESIOPs

Los Protocolos Inter-ORB Específicos del Entorno (ESIOPs, Environment-Specific Inter-ORB Protocols)proporcionan especificaciones para un conjunto de protocolos que se usarían para la interoperación con componentes que cuentan con una infraestructura de computación distribuida ya en uso.

2.7 Evolución de CORBA

Desde que se publicó la primera versión de CORBA (versión 1.0, en 1991), esta arquitectura ha ido evolucionando como cualquier tecnología o estándar que tiene que ir siendo adaptado a las nuevas necesidades de los usuarios. En CORBA 1.0 no se especifica la implementación del ORB, por lo que surgen ORBs basados en protocolos propietarios.

Pero a medida que las aplicaciones crecían, se crea la necesidad de interaccionar con otras aplicaciones que utilizan otros ORBs. Por este motivo en la versión CORBA 2.0, que fue publicada en 1995, se plantea la interoperabilidad entre ORBs mediante protocolos estándares y se especifican los protocolos IIOP (encima de TCP/IP) y ESIOP (para otros, por ejemplo DCE), que no sólo sirven para la comunicación entre ORBs distintos sino que también da la posibilidad de implementar el ORB con IIOP (o ESIOP). Se añadió el interfaz del Esqueleto Dinámico y extensiones al almacén de interfaces, así como la posibilidad de hacer callbacks, donde el cliente actúa como servidor.

En la versión 2.2 publicada en 1998, las actualizaciones más notables son la definición del Adaptador de Objetos Portable y el mapeado de IDL a JAVA. Las nuevas características que serán incluidas en la nueva versión (3.0) son la mensajería en CORBA y objetos por valor [Vin98].

2.8 Otras Plataformas Distribuidas

Aunque CORBA ha sido utilizada como base para este trabajo, hay un conjunto de tecnologías para el desarrollo de sistemas distribuidos que son dignas de mención. Las más importantes en la actualidad son DCE, RMI y DCOM. A continuación se comentan brevemente.

2.8.1 DCE

El entorno de computación distribuida (DCE, Distributed Computing Environment) [The96] es un conjunto de servicios de software desarrollado por The Open Group, un consorcio de usuarios y vendedores que trabajan juntos para avanzar en la tecnología de sistemas abiertos y que permite el desarrollo de aplicaciones distribuidas para sistemas heterogéneos.

DCE fue diseñado para operar independientemente del sistema operativo y de la tecnología de red que usan las aplicaciones. Por lo tanto permite a clientes y servidores interaccionar en cualquier tipo de entorno.

Esta tecnología se compone de servicios que operan por encima del sistema operativo. Estos servicios son los siguientes:

- Llamadas a procedimientos remotos: Facilitan la interacción entre clientes y servidores, permitiendo el acceso a recursos remotos.
- Servicio de seguridad: Autentifica las identidades de los usuarios, autoriza el acceso a los recursos en un sistema distribuido, y proporciona gestión de cuentas de usuarios y servidores.
- Servicio de directorio: Proporciona un modelo de nombrado único en entornos distribuidos.
- Servicio de tiempo: Permite la sincronización de relojes del sistema a través de la red.
- Servicio de hilos: Proporciona la capacidad de ejecutar múltiples hilos.
- Servicio de ficheros distribuidos: Proporciona el acceso a ficheros a través de la red.

En la figura 2.3 se muestra el esquema de las distintas partes de DCE, aunque el esquema en niveles no es exacto, ya que hay dependencias recursivas. Por ejemplo, el servicio de directorio utiliza las llamadas a procedimientos remotos para la comunicación entre sus diversos servidores, y a su vez, el paquete de llamadas a procedimientos remotos, utiliza el servicio de directorios para localizar al destino.

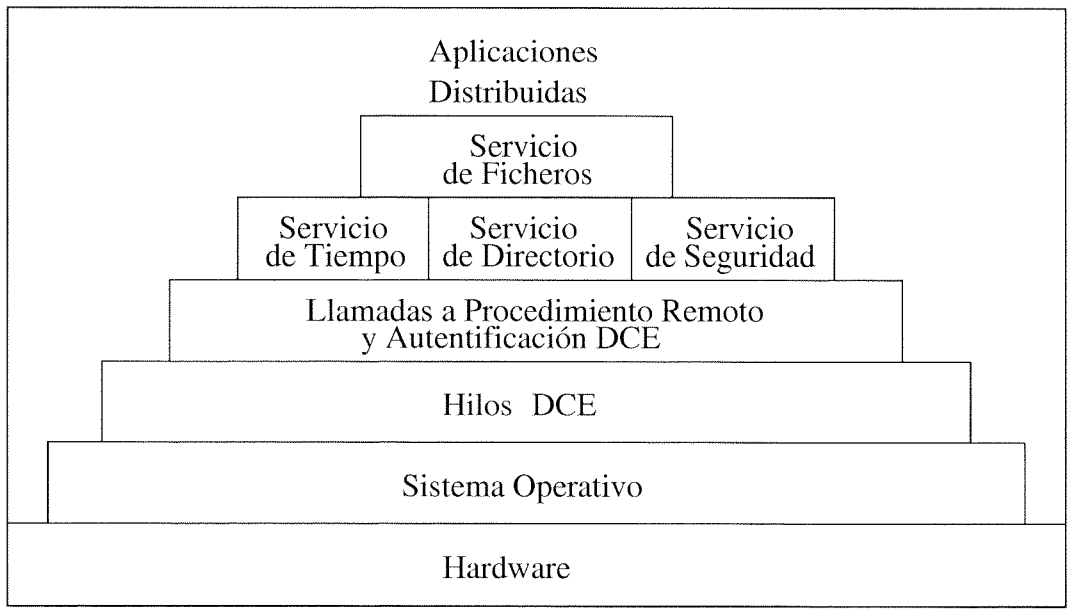


Figura 2.3: Componentes de DCE.

El entorno que proporciona, permite integrar servicios claves en aplicaciones distribuidas, aislando al desarrollador de la complejidad de la red subyacente y sus mecanismos de transporte.

DCE proporciona una base para la computación distribuida orientada a objetos. De hecho, los servicios de DCE han sido utilizados para la implementación de diversos ORB's, para lograr la interoperabilidad, y es usado por Microsoft como base para la comunicación en DCOM.

2.8.2 RMI

Invocación de Métodos Remotos (RMI, Remote Method Invocation) [Sun97] es un modelo de objetos distribuidos para la plataforma Java. RMI extiende el modelo de objetos de Java de forma que los métodos de los objetos pueden ser invocados en diferentes máquinas virtuales a través de una red, y permite que objetos reales puedan ser pasados como argumentos y valores de retorno durante la invocación de métodos.

Java RMI usa la serialización de objetos para convertir un objeto en un flujo de bytes para ser transportados por la red. Cualquier tipo de objeto Java puede ser pasado durante la invocación, incluyendo tipos primitivos, clases del núcleo, clases definidas por el usuario y JavaBeans. Java RMI podría ser descrito como una evolución natural de las llamadas a procedimiento remoto (RPC, Remote Procedure Call) para adaptarlas al paradigma orientado a objetos.

Debido a que RMI puede resolver dinámicamente invocaciones de métodos fuera de los límites de una Máquina virtual, proporciona un entorno distribuido orientado a objetos. Los desarrolladores pueden implementar aplicaciones diseñadas mediante métodos clásicos orientados a objetos para programación distribuida de la misma forma que si fuera local.

Con RMI se trabaja en un modelo de objetos simple, ya que está pensado para operar en el dominio Java. No permite por lo tanto la utilización de modelos de objetos múltiples como se permite en el caso de CORBA. Esto conlleva no obstante una disminución de la complejidad ya que no requiere ningún mapeado desde un lenguaje neutral de especificación de interfaz, y la sintaxis de la invocaciones a métodos remotos es prácticamente la misma a la de las invocaciones remotas.

Estas características hacen que la programación RMI sea más simple y adecuada para aquellas aplicaciones cliente/servidor completamente desarrolladas en Java, pero no para aquellas en las que los distintos componentes tienen que estar abiertos a peticiones de componentes escritos en otros lenguajes de programación.

2.8.3 ActiveX/DCOM

ActiveX/DCOM, más conocido como OLE (Object Linking and Embedding, Objetos enlazados e integrados), es un modelo propietario de Microsoft para tener componentes software en Windows de Microsoft, aunque está siendo extendido para soportar otras plataformas.

La tecnología OLE de Microsoft apareció en 1990 para proporcionar la capacidad de cortar y pegar en Windows. Fue extendido para producir OLE2, que permitía una comunicación más general entre aplicaciones Windows, permitiendo la integración de un tipo de documento en otro, por ejemplo, de una hoja de cálculo de Excel en un documento de Word. OLE2 permitió además realizar Drag and Drop para llevar a una ventana un objeto localizado en otra ventana.

Al nuevo modelo de comunicación utilizado en OLE2 se le dio el nombre de COM (Component Object Model). En COM sólo se permite la comunicación entre procesos

de la misma máquina, por eso fue extendido para poder comunicar procesos en distintas máquinas. A esta extensión se le conoce como DCOM (Distributed Component Object Model) [Chu97].

DCOM puede ser visto como una infraestructura de propósito general para la construcción de aplicaciones software basadas en componentes. Como CORBA, DCOM proporciona un conjunto de servicios que permite a distintos componentes colaborar de forma inteligente.

DCOM proporciona un protocolo de negociación de interfaz que permite a los clientes adquirir punteros a las interfaces de los componentes soportados en tiempo de ejecución.

A finales de 1994, Microsoft introdujo los Controles Clientes (OCXs, Custom Controls) que tienen un conjunto de interfaces predefinidas. Un OCX es un componente empaquetado de granularidad media bien definido. Un OCX no puede integrar otros componentes, para esto se usan los contenedores OLE (containers). De esta forma, un OCX es un conjunto de contratos entre un componente visual y su contenedor. Estos contratos deben ser implementados por el proveedor OCX. COM permite acceder a interfaces OLE existentes, personalizarlas mediante OCXs o crear nuevas.

Como CORBA, DCOM utiliza un lenguaje de definición de interfaz (IDL,Interface Definition Language) para la declaración de todas las interfaces, permitiendo la separación entre la interfaz de un objeto y su implementación. IDL de Microsoft está basado en DCE. Los programadores pueden escribir sus propias interfaces usando este lenguaje.

DCOM también proporciona un lenguaje de descripción de objetos (ODL, Object Description Language) para describir estos interfaces en una librería de tipos. Esta librería es equivalente al almacén de interfaces de CORBA.

DCOM no soporta herencia múltiple en las especificaciones IDL, sin embargo, un componente puede soportar interfaces múltiples, logrando la reutilización por medio de dos mecanismos que permiten que componentes externos representen a componentes internos en vez de utilizar la herencia. Estos dos mecanismos son:

- La contención(inclusión)/delegación: Un objeto externo contiene a uno o varios objetos internos. Cuando un cliente realiza una llamada a un método de la interfaz del objeto interno, es el objeto externo el que se encarga de redirigir esta llamada, delegando esta función en el objeto interno como se muestra en la figura 2.4.

- La agregación: En este caso, el objeto externo en vez de redirigir la llamada al

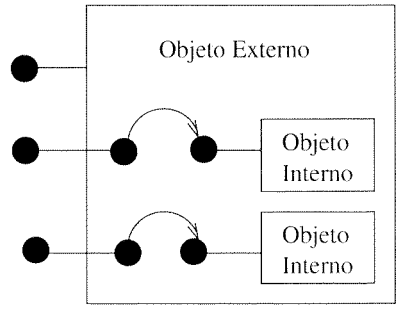


Figura 2.4: Contención/Delegación

objeto interno, expone directamente a sus clientes los punteros a las interfaces de los objetos internos, de forma que pueden llamar directamente a los métodos de los objetos internos sin la intervención del objeto externo. Esta idea se refleja en la figura 2.5

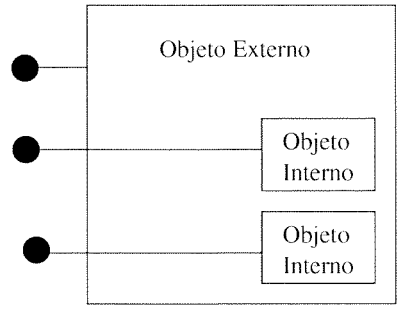


Figura 2.5: Agregación

Los objetos DCOM no son objetos en el sentido OO ya que no mantienen su estado entre conexiones. Una interfaz DCOM no puede ser instanciada para crear un objeto único. Una interfaz es un grupo de funciones relacionadas y el cliente dispondrá de un puntero para acceder a una tabla de punteros a las funciones de la interfaz, conocida como tabla virtual o vtabla. El esquema de la tabla virtual se muestra en la figura 2.6. Las funciones apuntadas son la implementación de los métodos del objeto. Cada objeto DCOM tiene una o más vtablas que definen el contrato entre la implementación del objeto y sus clientes.

Como CORBA, DCOM proporciona tanto interfaces estáticas como dinámicas. La Librería de Tipos en DCOM hace las funciones del Almacén de Interfaces en CORBA. En la Librería de Tipos se anotan las descripciones ODL de los objetos, incluyendo sus interfaces y parámetros. Los clientes pueden acceder a esta Librería de Tipos para descubrir dinámicamente las interfaces soportadas por un objeto y los parámetros que necesita para invocar a un método particular. DCOM también proporciona funciones

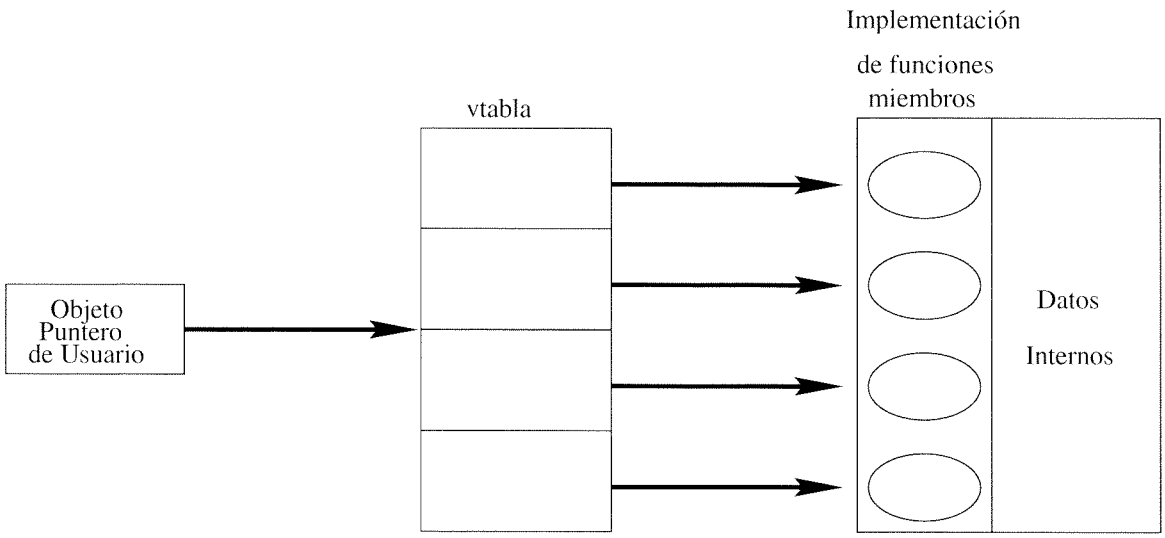


Figura 2.6: Tabla virtual

similares al Almacén de Implementaciones de CORBA, mediante un registro y algunos servicios de búsqueda de objetos.

Acceso Local y Remoto

Dependiendo de la relación con el proceso cliente, en DCOM los servidores pueden ser implementados de varias formas:

- Servidores en el proceso: Estos servidores se ejecutan en el mismo espacio de proceso que sus clientes. Son normalmente librerías de enlace dinámico (DLLs, Dynamic Link Libraries) que son cargadas en el proceso cliente.
- Servidores locales: Se ejecutan en un espacio diferente a sus clientes, pero en la misma máquina que éstos. Los clientes utilizan el mecanismo LRPC (Lightweight Remote Procedure Call) de DCOM para comunicar con el servidor local.
- Servidores remotos: Se ejecutan en una máquina remota que puede incluso tener un sistema operativo diferente. En este caso, se utiliza un mecanismo similar a las RPCs de DCE para comunicar con el servidor.

DCOM oculta al cliente las diferencias entre las llamadas a métodos de servidores de distinta naturaleza.

Desde el punto de vista del cliente se utiliza un puntero a interfaz independientemente de que el servidor esté en el mismo proceso, en un proceso diferente en la misma

máquina o en una máquina remota. Al realizar la llamada a un método, ésta se hace directamente si el servidor está en el mismo proceso. En caso contrario, interviene un elemento proxy generado por DCOM. Este elemento proxy es el que realiza la llamada a procedimiento mediante el mecanismo adecuado, bien LRPC o RPC.

Desde el punto de vista del servidor, la llamada la realiza el propio cliente en caso de estar en el mismo proceso, o un "stub" o soporte que se encarga de gestionar la LRPC o RPC y la llamada al método del servidor en caso de estar en un proceso diferente.

Este mecanismo es similar al que utiliza CORBA para proporcionar transparencia de localización, usando el "stub" o soporte en el lado cliente y el esqueleto en el lado servidor.

Interoperatividad CORBA/DCOM

En la especificación CORBA se incluye la especificación de un puente que proporciona interoperabilidad entre CORBA y DCOM. Este puente incluye los siguientes servicios:

- Mapeado de invocación de métodos.
- Mapeado de tipos de datos.
- Mapeado entre CORBA IDL y OLE ODL.
- Mapeado de excepciones.
- Mapeado de inicialización.
- Mapeado de fábrica de objetos.
- Mapeado de registro.
- Mapeado de dominios de seguridad.

El propósito de esta especificación es lograr la interoperación entre objetos DCOM y objetos CORBA. De esta forma, un cliente CORBA puede realizar una llamada a un método de un objeto DCOM y un cliente DCOM puede realizar una llamada a un método de un objeto CORBA.

Capítulo 3

Estado del Arte de la Gestión de Sistemas

En este capítulo se da una visión general de la gestión de sistemas, mencionando la importancia que ha cobrado en los últimos años. Se plantea el problema de las soluciones propietarias que han dejado paso a modelos estándares para la gestión como son los modelos de gestión ISO e Internet.

Se realiza una descripción general de TMN, ODMA y TINA que son arquitecturas más avanzadas que son utilizadas en el campo de las telecomunicaciones.

Se resume también en este capítulo la influencia que ha tenido CORBA en el entorno de gestión. Se finaliza comentando otras tendencias.

3.1 Importancia de la Gestión de Sistemas

Debido al incremento del número de componentes que conforman un sistema distribuido surge la necesidad de monitorizar y controlar el funcionamiento de tales componentes así como de las redes de comunicación que posibilitan la interconexión entre ellos.

Uno de los objetivos principales de la monitorización es la determinación del nivel de funcionamiento correcto o condiciones dentro de las cuales es posible definir un estado de operación normal.

Mediante la utilización de un conjunto básico de herramientas que sean capaces de

controlar tanto las redes como los sistemas que intervienen, es posible operar de forma activa de cara a resolver incidencias, excepciones y escenarios de operación anormales, así como configurar los parámetros de funcionamiento de dichos sistemas. Esto último permite alterar las condiciones de funcionamiento cuando previamente se hayan detectado anomalías.

Por otro lado, los componentes individuales que conforman el sistema distribuido pueden tener la capacidad de generar notificaciones que revelen situaciones excepcionales, posibles fallos o informes de estado con una cierta periodicidad.

La Gestión de Redes y Sistemas es una Aplicación Distribuida en la que intervienen un número determinado de Entidades de Aplicación. Algunas de estas entidades dan soporte, en un determinado sistema, a aplicaciones de monitorización y de control sobre el resto de los elementos de la red y del sistema, normalmente denominadas Gestores.

En el otro extremo de la Aplicación de Gestión están los elementos que van a ser monitorizados y controlados por las anteriores. A estos componentes se les denomina normalmente Agentes.

El modelo de interconexión de sistemas abiertos (OSI) define 5 áreas funcionales que especifican los fines de la gestión, son aspectos en los que se requiere la gestión y son los siguientes:

- Gestión de fallos: comprende la detección, el aislamiento y la corrección de fallos. Los fallos hacen que los sistemas dejen de satisfacer sus objetivos operacionales; pueden ser persistentes o transitorios. Los errores se manifiestan como sucesos particulares en la operación del sistema. La detección de errores proporciona una capacidad para reconocer fallos. La gestión de fallos incluye funciones para:
  - Mantener cuadernos de error.
  - Aceptar y reaccionar a notificaciones.
  - Rastrear e identificar fallos.
  - Secuencias de pruebas.
  - Eliminar fallos.
- Gestión de contabilidad: permite establecer cargos (o tasas) por el uso de recursos e identificar costos correspondientes a la utilización de esos recursos. La gestión de contabilidad incluye funciones para:
  - Informar de costos o recursos consumidos.
  - Mantener calendarios de tarifas.
  - Permitir la combinación de costos.

- Gestión de configuración: identifica y ejerce control sobre los sistemas gestionados, toma datos referentes a éstos y proporciona datos con el fin de preparar, inicializar, poner en marcha, y tener en cuenta la operación continua y la terminación de servicios. Entre las funciones de la gestión de configuración se encuentran:
  - Establecer parámetros.
  - Iniciar y cerrar objetos gestionados.
  - Reunir información.
  - Obtener anuncios de cambios.
  - Cambiar la configuración.
- Gestión de (calidad de) funcionamiento: permite evaluar el comportamiento de recursos y la efectividad de las actividades. La Gestión de funcionamiento trata entre otras cosas de:
  - Información estadística.
  - Cuadernos de historiales.
  - Determinar el rendimiento.
  - Cambiar los modos de operación.
- Gestión de seguridad: tiene por finalidad soportar la aplicación de políticas de seguridad. Se pueden nombrar como parte de la gestión de seguridad las siguientes funciones:
  - Creación, supresión y control de servicios y mecanismos de seguridad.
  - Distribución de la información relativa a la seguridad.
  - Señalización de sucesos relacionados con la seguridad.

En los siguientes apartados se comentan distintas soluciones y paradigmas de gestión de una forma muy reducida, en [Mar98] se puede encontrar un resumen más exhaustivo de las distintos paradigmas de gestión.

3.2 Soluciones Propietarias

Antes de la adopción de estándares para la integración de la gestión de sistemas, cada fabricante propone su propia solución, surgiendo de esta forma una gran variedad de protocolos con esquemas de funcionamiento diferentes e incompatibles entre sí.

En sistemas heterogéneos donde hay una diversidad de componentes, probablemente de diferentes fabricantes, la adopción de soluciones propietarias para la gestión de estos elementos es inviable. Las distintas soluciones propietarias conllevarían el tener en una misma empresa distintos subsistemas de gestión, con el consiguiente gasto de recursos en equipos y personal adecuado para cada uno de estos subsistemas.

Por este motivo las distintas soluciones propietarias han ido dejando paso a estándares como los comentados en los subapartados siguientes o a modelos que permiten la integración de estos protocolos propietarios en sistemas de gestión no propietarios.

3.3 Modelo de Gestión Internet

El modelo de gestión Internet está basado en el esquema cliente/servidor. Los servidores son llamados agentes y los clientes son los gestores. El agente interacciona con el sistema para obtener y manipular la información de gestión. El agente expone la información de gestión comunicándose con los gestores por medio del protocolo SNMP (Simple Network Management Protocol). Este protocolo está estandarizado por IETF (Internet Engenieering Task Force) en la RFC 1157 [RFC90b].

Los agentes están localizados en cada entidad del sistema y pueden ser monitorizados. El gestor es una aplicación software que se ejecuta en un centro de operación.

Las operaciones de gestión son representadas como alteraciones o inspecciones de objetos, estos objetos son las variables de la MIB. Las variables pueden ser obtenidas mediante la operación Get o alteradas mediante la operación Set. La monitorización se realiza básicamente mediante un sondeo o polling para obtener información. Sin embargo, el agente puede enviar mensajes no solicitados mediante la operación Trap (aunque no está normalizado el uso de esta operación).

3.3.1 Protocolo SNMP

El protocolo SNMP es un protocolo de solicitud/respuesta. Las operaciones que contempla el protocolo son las mostradas en la tabla 3.1. Se pueden solicitar Operaciones más complejas por medio de efectos laterales de la operación Set.

En general, los mensajes Get, GetNext y Set son generados por un gestor, y los mensajes GetResponse y Traps son generados por agentes. El gestor envía solicitudes SNMP al agente para obtener información mediante Get, GetNext y/o para cambiar

Mensaje	Descripción
Get Request	Obtener variable(s) de la MIB de forma directa
GetNext Request	Interroga variable(s) de la MIB según el orden lexicográfico de las instancias
Get Response	Respuesta a un Get, Get_Next o Set request
Set Request	Solicitud para cambiar variable(s) de la MIB a los valores proporcionados
Trap	Un aviso indicando que algo ha ocurrido (arranque de un sistema, caída de un enlace, fallo de identificación, etc.) Un tipo de trap específico de empresas permite a los vendedores realizar extensiones

Tabla 3.1: Mensajes SNMP

el valor de algunas variables mediante Set. El agente responde a cada una de estas solicitudes con GetResponse. Normalmente el gestor SNMP estará realizando las solicitudes anteriores al agente, de forma que éste actúa de forma pasiva. Sólo en algunos casos, el agente tomará la iniciativa enviando un trap. Si el gestor recibe un trap puede efectuar la acción apropiada para obtener más información.

3.3.2 Modelo de Información de Gestión

El protocolo SNMP no especifica qué datos, objetos o variables son usados para la gestión, o cómo es representada la información de gestión. En vez de eso, el estándar referente a la Estructura de la Información de Gestión (SMI, Structure of Management Information) [RFC90a] proporciona un marco sintáctico para definir la información de gestión. La Base de Información de Gestión (MIB, Management Information Base) [RFC91a] es construida a partir de un subconjunto simple de tipos de datos de la sintaxis abstracta ASN.1 apoyándose en el SMI. Los elementos de información de gestión en la MIB son definidos como tipos de objetos con las siguientes propiedades:

- Descriptor de objeto: Un nombre inteligible para los humanos.
- Identificador de objeto: Un identificador ASN.1.
- Sintaxis: Uno de los siguientes tipos ASN.1 definidos: INTEGER, OCTET STRING, NULL, OBJECT IDENTIFIER, IpAddress, Counter, Gauge, TimeTicks, Opaque.

Estructuras más complejas, por ejemplo tablas, se pueden crear por medio de agrupaciones en registros de tipos de objetos.

Otro estándar Internet conocido como MIB II [RFC91b] define la información de gestión relevante para gestionar redes TCP/IP. Los objetivos de diseño de la MIB II se pueden resumir como los siguientes:

- Mantener la independencia entre la información de gestión y el protocolo de gestión.
- Definir un conjunto básico de información requerida para gestionar redes TCP/IP.
- Permitir que se añada información específica de cada organización.

SNMP usa el árbol de registro ISO definido en la recomendación X.660 [ITU92a] para nombrar las variables de la MIB. A cada tipo de objeto se le asigna un identificador ASN.1. El direccionamiento de los datos en la MIB se realiza usando el identificador de objeto asignado al tipo de objeto como una dirección. Como el direccionamiento de los datos se realiza a un solo nivel, no se puede realizar por este método el direccionamiento a varias instancias de un mismo tipo. El direccionamiento de múltiples instancias de un mismo tipo de dato se realiza agrupando las instancias en columnas de una tabla conceptual y usando los valores de algunas de estas columnas como direcciones adicionales asociadas. Excepto si la información es modelada como tablas en la MIB se entiende que existe una sola instancia del tipo de objeto en el agente.

Desde la primera versión de SNMP conocida como SNMPv1, han surgido posteriormente otras versiones. SNMPv2 añade al protocolo cuestiones de seguridad más avanzadas que las incluidas en la versión 1. Incluso ha sido propuesta la versión SNMPv3. Una revisión de todos los estándares relacionados con SNMP pueden ser encontrados en [Cek98].

3.4 Modelo de Gestión OSI

El modelo de gestión OSI [ITU97c] también responde al esquema cliente servidor, donde el cliente es el gestor y el servidor es el sistema gestionado. El sistema gestionado hace el papel de agente recibiendo operaciones a realizar sobre los objetos gestionados y encaminando notificaciones emitidas por los objetos. El gestor invoca operaciones de gestión y recibe notificaciones. Un mismo sistema puede desarrollar las funciones de gestor o de agente según el sistema con el que esté dialogando.

En el modelo de gestión OSI se pueden realizar operaciones sobre instancias de objetos y sus propiedades, incluyendo atributos.

El protocolo utilizado en este modelo es CMIP (Common Management Information Protocol) [ITU97a] que es más amplio pero también más complejo que SNMP. Este protocolo es utilizado sobre todo en el sector de las telecomunicaciones ya que fue adoptado por el ITU-T como las bases para el modelo TMN.

3.4.1 Protocolo CMIP

En el protocolo de gestión CMIP se define un conjunto más amplio de mensajes que en SNMP. En este protocolo se distinguen las operaciones sobre objetos de las operaciones sobre atributos. En el estándar CMIS (Common Management Information Service) [ITU97b] se describen los servicios que son realizados mediante el intercambio de PDUs CMIP. Los tipos de PDUs se describen en la tabla 3.2.

Mensaje	Descripción
EventReport	Informa de un suceso sobre un recurso gestionado a otro usuario del servicio CMIS
Get	Pide información de otro usuario del servicio CMIS
Set	Solicita la modificación de la información de gestión
Action	Solicita a otro usuario del servicio CMIS que realice una acción
Create	solicita a otro usuario del servicio CMIS que cree un objeto gestionado
Delete	Solicita a otro usuario del servicio CMIS que borre un objeto gestionado
CancelGet	Invocado para cancelar un Get realizado anteriormente

Tabla 3.2: Mensajes CMIP

La mayoría de los mensajes CMIP contienen los siguientes parámetros:

- El tipo de operación o notificación.
- Un identificador de invocación para la correlación entre la solicitud y la respuesta.

- La identidad de el/los objeto/s gestionado/s sobre el/los que se realiza la operación o es fuente de la notificación.
- Información específica del objeto o de la operación.

Las instancias de objetos son organizadas en una jerarquía de contenencia por sus nombres. La identidad de los objetos gestionados sobre los que se va a realizar la operación puede realizarse por una combinación de:

- Su tipo (clase de objeto gestionado).
- Su nombre (instancia de objeto gestionado).
- Su posición en la jerarquía de inclusión (ámbito).
- Un predicado refiriéndose a sus atributos (filtro).

Esto permite a un gestor afectar a varios objetos con una sola operación, y seleccionar objetos dependiendo de su estado dinámico.

En general, las solicitudes CMIP Get, Set, Create, Delete, Action y CancelGet son generadas por el sistema gestor, mientras que las respuestas y los EventReport son generados por el agente.

Un EventReport es una indicación no solicitada de que un suceso importante ha ocurrido en el sistema gestionado. Contiene una indicación del suceso e información asociada que puede ser útil para entender el suceso. CMIP soporta que los EventReport sean confirmados o no confirmados. El modelo de gestión OSI tiene una visión de la gestión dirigida por sucesos, por lo tanto se hace un gran uso de los EventReport.

El ámbito y el filtrado permiten que las operaciones CMIP sean aplicadas a todas la instancias de la MIB del agente. Así por ejemplo se pueden obtener todas las instancias y los valores de todos los atributos de un agente con una sola solicitud GET. El parámetro de filtrado define qué condiciones deben satisfacer los atributos para que la operación se lleve a cabo. El parámetro ámbito permite definir al solicitante en qué parte de la jerarquía de inclusión va a realizar la operación. Cuando una operación se aplica a varios objetos puede generar varias respuestas a una sola solicitud. Por ejemplo, la realización de un Get sobre un par de objetos generará una respuesta por cada uno de los objetos, cada respuesta conteniendo los valores de los atributos de cada objeto.

La capacidad de controlar la gestión en sí misma está también especificada como funciones de gestión. Estas funciones normalmente definen objetos que representan las

capacidades de control. Esto permite que el mismo protocolo y modelo de información de gestión sean usados para controlar los servicios de gestión, no se requiere un protocolo de gestión especializado. Por ejemplo, la función de notificación de eventos define un objeto discriminador de envío de eventos que permite a los gestores iniciar, configurar, suspender, reanudar, terminar el envío de eventos por parte del agente.

3.4.2 Modelo de Información de Gestión

El modelo de información de gestión OSI es un modelo orientado a objetos en el que se representan los recursos y sistemas del mundo real como objetos gestionados. Los objetos se caracterizan por:

- Las operaciones que aceptan.
- Las notificaciones que emiten.
- Los atributos que tienen disponibles.
- Los comportamientos que exhiben.

GDMO (Guidelines for the Definition of Managed Objects) [ITU92b] de OSI define unas plantillas para expresar estas propiedades. Los objetos que comparten especificaciones idénticas son agrupados en clases. Especificaciones de nuevas clases se pueden obtener a partir de las clases ya existentes usando la herencia. La nueva clase (subclase) hereda todas las especificaciones de la clase original (superclase), y añade nuevas especificaciones. El diseño orientado a objeto de la metodología OSI proporciona 4 funciones claves: encapsulación, estructura de clases de objetos, herencia y alomorfismo.

OSI normaliza y clasifica todas las posibles relaciones entre objetos, siendo una de ellas especialmente relevante en la estructura de objetos y la identificación de los mismos: la relación de inclusión o dependencia. En general, dos objetos están relacionados por inclusión cuando la existencia del objeto incluido (objeto subordinado), depende de la existencia del objeto incluyente (objeto superior). Se obtiene de esta forma, un árbol jerárquico de objetos.

La relación de inclusión puede ser indirecta, es decir, un objeto A depende indirectamente de otro C si tenemos un objeto intermedio B que depende directamente de C, con A dependiendo directamente de B; relación transitiva de dependencias. En general, se obtiene un árbol de dependencias de objetos con tantas ramificaciones como sean necesarias para una buena representación.

Los objetos, en general, suelen tener algún atributo en común. Por lo tanto, los conjuntos de atributos, acciones y notificaciones que forman el total de objetos de la MIB son conjuntos no disjuntos entre sí.

La identificación de objetos GDMO se realiza mediante el atributo específico "name". en dicho atributo se guarda el Nombre Distinguido Relativo (RDN, Relative Distinguished Name). Este nombre es único en el conjunto de objetos que dependen de un mismo objeto superior o padre. El objeto superior a un conjunto de objetos subordinados marca el contexto de gestión de dichos objetos subordinados. El contexto de gestión viene por tanto determinado por el Nombre completo o Distinguido (DN, Distinguished Name) del objeto superior. Se puede definir un nombre distinguido como la secuencia de nombres relativos desde el objeto en cuestión hasta el objeto "techo" del árbol de objetos. Una dirección de información mínima CMIS correspondería a la tupla formada por un DN del padre más un RDN del hijo.

En GDMO se puede definir el esquema de nominado posible de un conjunto de objetos mediante la definición de tipos de datos llamados "name binding" (vinculación de nombre). Cada vinculación expresa los tipos de objetos a los que debe pertenecer las instancias conformes a dicha vinculación.

3.5 TMN

El término Red de Gestión de Telecomunicaciones (TMN, Telecomunications Management Network) ha sido definido por el ITU-T en la Recomendación M.3010 [ITU96] como una red separada que conecta con la red de telecomunicaciones por distintos puntos. En esta recomendación se definen conceptos generales de la gestión TMN y se introducen distintas arquitecturas de gestión a diferentes niveles de abstracción:

- Arquitectura Funcional: Describe un conjunto de funciones de gestión.
- Arquitectura de Información: Describe conceptos que han sido adoptados del modelo de gestión OSI.
- Arquitectura Física: Define la forma en que las funciones de gestión pueden ser implementadas en equipos físicos.
- Arquitectura Lógica Nivelada: Incluye un modelo de gestión estructurado de acuerdo a diferentes responsabilidades.

A continuación se exponen brevemente cada una de estas arquitecturas.

3.5.1 Arquitectura Funcional

En la arquitectura funcional de TMN se definen cinco tipos diferentes de bloques funcionales:

- Bloque Funcional de Sistema de Operación (OSF, Operations System Functions): Procesa la información relacionada con la gestión de las telecomunicaciones con el propósito de monitorizar, coordinar y/o controlar las funciones de telecomunicación.

- Bloque Funcional de Mediación (MF, Mediation Functions): Actúa sobre la información que llega de los NEF y de los QAF para adaptarla, filtrarla y condensarla adecuándola al formato en que es esperada por los OSF.

- Bloque Funcional de Estación de Trabajo (WSF, Work Station Functions): Proporciona los medios para conectar al usuario con el sistema de operaciones. El propósito de esta función es garantizar la independencia de datos entre interfaces de usuario particulares. Los aspectos de interfaz hombre-máquina no están incluidos dentro de TMN, y por lo tanto caen fuera de este bloque funcional.

- Bloque Funcional de Elemento de Red (NEF, Network Element Functions): Se comunica con TMN con el propósito de ser monitorizado y/o controlado. Representa los aspectos de los recursos de telecomunicación de utilidad para la gestión de dicho recurso.

- Bloque Funcional de Adaptador Q (QAF, Q Adaptor Functions): se usa para conectar a TMN aquellos elementos de red que no soporten un interfaz TMN estándar. Su responsabilidad es realizar la correspondencia entre ambos interfaces.

Los dos primeros bloques (OSF y MF) están totalmente definidos en las recomendaciones TMN. De los tres tipos restantes (WSF, NEF y QAF) sólo parte de estos bloques de función están especificados por TMN.

En esta arquitectura se introduce el concepto de puntos de referencia que sirven para delimitar los bloques funcionales. Se identifican 5 tipos de puntos de referencia (q, f y x, completamente descritos en TMN y g y m, sólo parcialmente). El modelo de referencia funcional de TMN se muestra en la figura 3.1 , donde se aprecia la interconexión de los distintos bloques funcionales por medio de los distintos puntos de referencia.

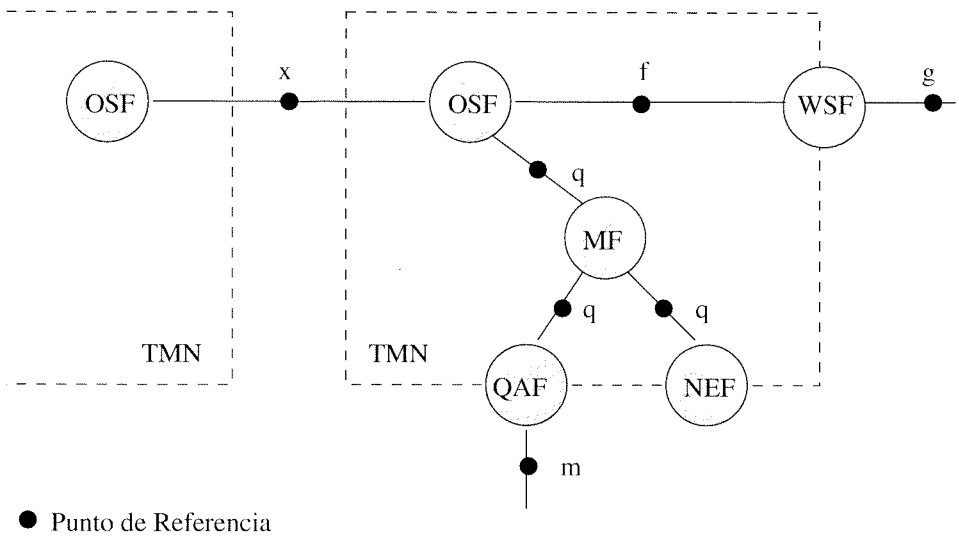


Figura 3.1: Bloques funcionales y puntos de referencia de TMN.

3.5.2 Arquitectura de Información

La arquitectura de información de TMN define el formato de la información que se transmite entre los bloques funcionales. Esta arquitectura está basada en el modelo de información de gestión de OSI. Usa una aproximación orientada a objetos. De acuerdo a este modelo, los objetos gestionados se describen en términos de:

- Atributos, que son propiedades o características del objeto.
- Operaciones, que son realizadas sobre los objetos.
- Comportamiento, que exhibe el objeto en respuesta a las operaciones realizadas.
- Notificaciones, que son emitidas por el objeto.

Los objetos gestionados residen en sistemas gestionados que incluyen funciones de agente para comunicar con el gestor. TMN usa el mismo concepto gestor-agente que OSI.

3.5.3 Arquitectura Física

La arquitectura física de TMN define cómo las funciones de gestión, que son definidas en la arquitectura funcional, se pueden implementar en equipos físicos. Esta arquitectura

muestra el mapeado de bloques de función en bloques de construcción (equipos físicos) y de puntos de referencia en interfaces.

3.5.4 Arquitectura Lógica Nivelada

Se incluye un modelo que muestra como la gestión puede ser estructurada de acuerdo a diferentes responsabilidades. Se presenta una jerarquía descrita en términos de niveles de gestión. La idea subyacente a esta arquitectura fue descrita en 1989 por BT como parte de su Arquitectura de Red Abierta (ONA, Open Network Architecture).

Para abordar la complejidad que conlleva la gestión, la funcionalidad de gestión junto con su información asociada se puede descomponer en un conjunto de niveles lógicos. Generalmente la descomposición de estas funcionalidades dan lugar a los siguientes niveles mostrados en la figura 3.2:

- Nivel de Gestión de Elemento: Trata con funciones de gestión específicas del fabricante y oculta estas funciones al nivel superior, es decir, el Nivel de Gestión de Red.
- Nivel de Gestión de Red: Su responsabilidad es gestionar las funciones relacionadas a la interacción entre múltiples piezas de equipamiento.
- Nivel de Gestión de Servicio: Se tratan en este nivel aspectos que pueden ser directamente observados por los usuarios de la red de telecomunicación (tanto usuarios finales como otros proveedores de servicio).
- Nivel de Gestión de Mercado: Es responsable de la gestión de la empresa completa. Su ámbito no se limita a la gestión de las comunicaciones, sino que es más amplio, incluyendo además gestión estratégica y táctica.

3.6 ODMA

La Arquitectura de Gestión Distribuida y Abierta (ODMA, Open Distributed Management Arquitecture) [UT97] extiende el modelo de gestión OSI, y por tanto la arquitectura TMN, con el Modelo de Referencia de OSI para el Procesamiento Distribuido y Abierto (RM-ODP). Por lo tanto, ODMA se considera como una interpretación específica del modelo de referencia ODP a efectos de gestión, introduciendo términos de carácter general para la gestión distribuida y abierta.

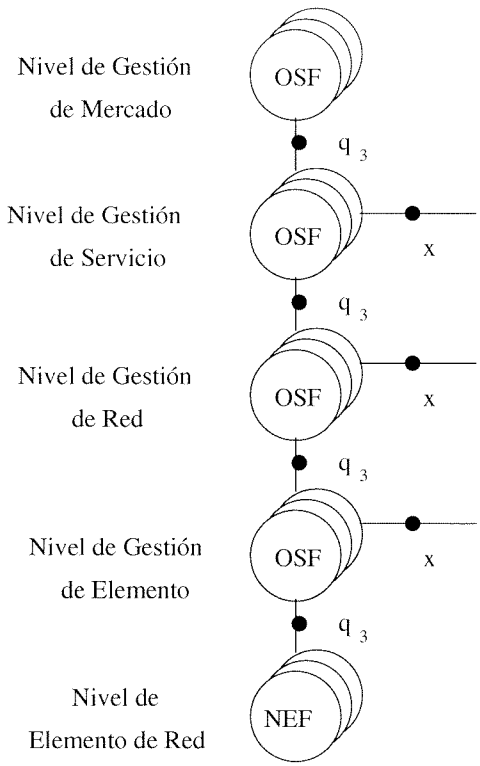


Figura 3.2: Arquitectura Lógica Nivelada.

Con ODMA se dispone de una arquitectura para la especificación y desarrollo de la gestión de sistemas vista como una aplicación distribuida y abierta así como de la gestión de aplicaciones distribuidas y abiertas. En este marco arquitectural, la gestión será distribuida y abierta, lo que significa que se tiene distribución de la actividad gestora, gestión de aplicaciones distribuidas y gestión de recursos que pueden estar distribuidos.

En ODMA no se habla de agente o gestor como en los modelos tradicionales, sino que siguiendo con RM-ODP, se hace referencia a objetos de gestión computacionales en su rol de gestionado o en su rol de gestor.

En la sección correspondiente al soporte de gestión OSI para ODMA se establece la relación entre conceptos de gestión OSI actuales y conceptos ODMA, ampliando las normas de gestión de sistemas actuales para dar cobertura a la distribución de las actividades de gestión y la distribución de los recursos que han de ser gestionados.

3.7 TINA

La arquitectura TINA (Telecomunications Information Networking architecture) [TIN97] ha sido introducida por TINA-C (TINA-Consortium) creado en 1993 para colaborar en la definición de una arquitectura común.

TINA define una arquitectura software para sistemas de telecomunicaciones. En esta arquitectura se definen conceptos y principios que son aplicados en la especificación, diseño, implementación, instalación, ejecución y operación del software para sistemas de telecomunicación. Está basada en dos principios fundamentales: las técnicas de computación distribuida y en las técnicas del software orientado a objetos.

Las aplicaciones de telecomunicaciones se diseñan y se implementan como un conjunto de objetos que interaccionan entre sí. El Entorno de Procesamiento Distribuido (DPE, Distributed Processing Environment) es el software responsable del soporte de la ejecución distribuida de estas aplicaciones.

Como se ve en la figura 3.3 el DPE está asentado sobre el Entorno de Comunicaciones y Computación Nativa (NCCE, Native Computing and Communications Environment). Este entorno está compuesto por un conjunto de nodos de computación interconectados, donde cada nodo puede soportar tecnología hardware y software diferente.

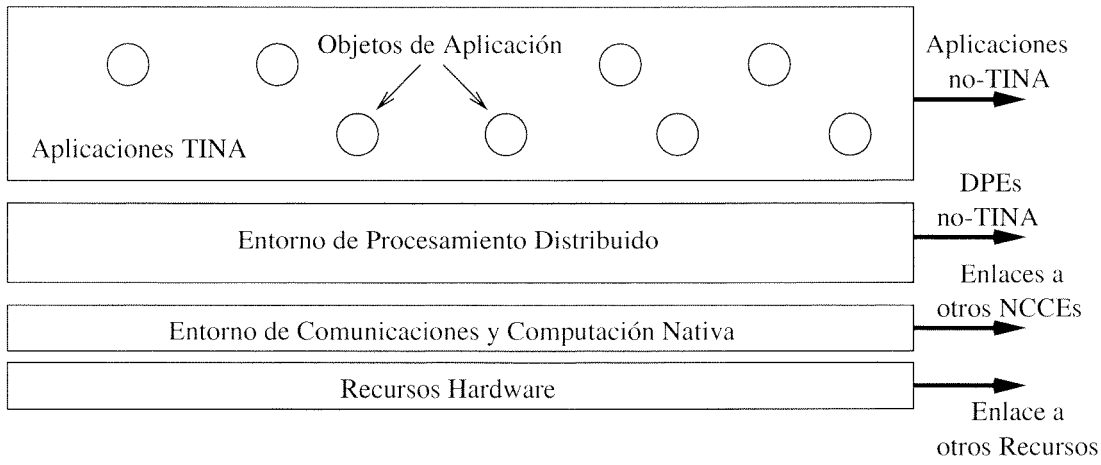


Figura 3.3: Estructura Básica en un Entorno TINA.

En el nivel inferior se encuentran los recursos hardware tales como procesadores, memoria y dispositivos de comunicación.

TINA realiza una separación lógica entre las aplicaciones de alto nivel y el entorno en el que se ejecutan (el DPE). También realiza una separación en las aplicaciones en

la parte específica de servicio y la parte de control y gestión genérica.

TINA está dividida en las siguientes arquitecturas:

- Arquitectura Computacional: Define un conjunto de conceptos y principios para el diseño y la construcción de software distribuido y el entorno de soporte del software. Los conceptos definidos en esta arquitectura están basados en RM-ODP.
- Arquitectura de Red: Define un conjunto de conceptos y principios para el diseño, especificación, implementación y gestión de redes de transporte.
- Arquitectura de Servicio: Define un conjunto de conceptos y principios para el diseño, especificación, implementación y gestión de servicios de telecomunicaciones.
- Arquitectura de Gestión: Define un conjunto de conceptos y principios para el diseño, especificación e implementación de sistemas software que son usados para gestionar los servicios, recursos, software y tecnologías subyacentes.

3.8 Influencia de CORBA en Gestión de Sistemas

Debido a las características de distribución y orientación a objetos que son aplicables en las distintas tecnologías desarrolladas para la gestión de sistemas, se han visto fuertemente influidas por la arquitectura CORBA, de forma que esta arquitectura no sólo ha sido extensamente aceptada en la comunidad de aplicaciones distribuidas sino que también está siendo gradualmente adoptada por la comunidad de gestión. Esto ha dado lugar a esquemas mixtos donde conviven técnicas de gestión tradicionales y CORBA. En los sistemas presentados en [Ase98, Kel98, Val99] se considera la convivencia entre CORBA y SNMP, mientras que en [Gen97, Gen98] se propone la integración del modelo de gestión OSI con los objetos distribuidos de CORBA.

En [Ram99] se ha realizado el diseño y la implementación de un Agente Proxy entre SNMP y CORBA basado en el esquema propuesto en [Ase98] que permite realizar peticiones CORBA a un agente SNMP.

En [Góm00] se ha complementado el trabajo anterior con el diseño y la implementación de un Gestor SNMP basado en CORBA que utiliza el Agente Proxy mencionado anteriormente para gestionar agentes SNMP.

Las arquitecturas anteriormente comentadas, tanto ODMA como TINA se han visto afectadas también por la arquitectura CORBA.

En la enmienda 1 de Junio del 98 de la recomendación correspondiente a ODMA [UT98] se ha incluido una sección con el soporte CORBA para ODMA. Se describe como la infraestructura de proceso distribuido adoptada por la OMG puede ser utilizada para el desarrollo de sistemas ODMA. Se amplían los conceptos ODMA con el término objeto gestionado en CORBA que se utiliza para indicar un objeto computacional gestionado en un entorno CORBA.

Por otro lado, TINA-C ha adoptado CORBA para su Entorno de Procesamiento Distribuido desde 1996.

En el capítulo 5 "Modelo de Gestión para Sistemas de Fabricación" se presenta un modelo básico de un sistema de gestión basado en CORBA.

3.8.1 Unión de Gestión Inter-Dominio

La Unión de Gestión Inter-Dominio (JIDM, Joint Inter-Domain Management) [Ope97, OMG98] es una tecnología que define cómo componentes de gestión basados en OSI y SNMP pueden interoperar con componentes basados en CORBA. Esta tecnología ha sido adoptada tanto por el Grupo de Gestión de Objetos (OMG), como por el Grupo Abierto (The Open Group) como por el Foro de TeleGestión (TeleManagement Forum, anteriormente NMF).

La especificación JIDM se centra en la definición de algoritmos de traducción entre el modelo de información de OSI o SNMP y especificaciones CORBA y en la definición de interfaces CORBA permitiendo todas las interacciones posibles en CMIP y SNMP. Los escenarios que son permitidos por estas especificaciones se muestran en la figura 3.4.

Para lograr la integración entre todas estas disciplinas de gestión, es necesario proporcionar interconexión entre las diferentes tecnologías que son empleadas. De esta forma, se permite introducir la tecnología aplicada a un dominio en otro, por ejemplo, se puede lograr integrar tecnología CORBA en sistemas de gestión basados en OSI o SNMP.

El JIDM ha generado el documento "Inter-domain Management: Specification Translation" [Ope97]. En este documento de especificación de traducción, se aborda entre otras cosas:

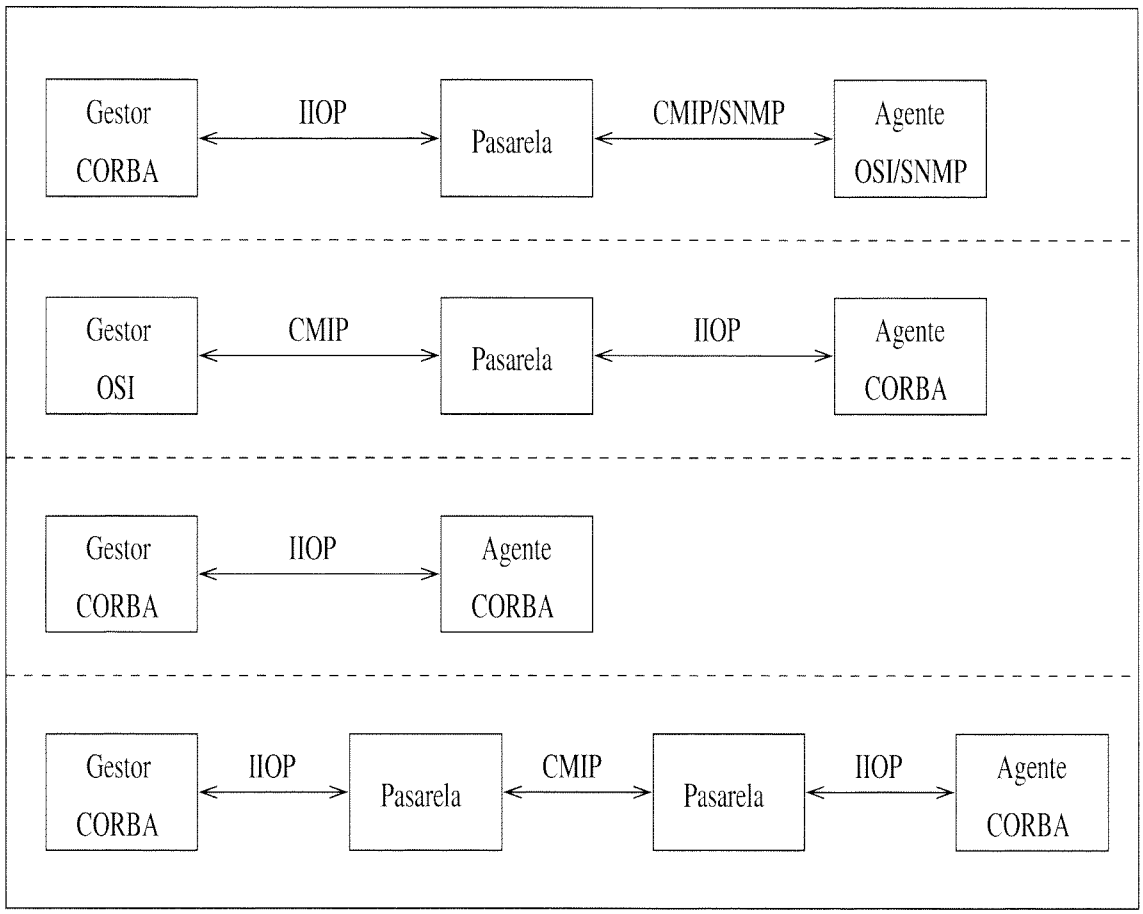


Figura 3.4: Escenarios JIDM.

- Traducción de ASN.1 a OMG IDL
- Traducción de GDMO a OMG IDL
- Traducción de OMG IDL a GDMO
- Traducción de SNMP a OMG IDL
- Traducción de OMG IDL a SNMP (No proporcionada actualmente)

Para este trabajo de tesis se ha tomado como referencia el primer punto mencionado, es decir, la traducción de ASN.1 a OMG IDL para llevar a cabo la traducción de MMS a IDL.

3.9 Otras Tendencias

Se exponen en este apartado otras tendencias actuales utilizadas en gestión de sistemas. Se van a mencionar JMX, WBEM, los sistemas de Código Móvil y los Agentes Inteligentes, aunque hay que considerar que a veces unas tecnologías influyen o son tomadas como base para otras, de forma que los límites entre ellas a veces no están demasiado claros.

3.9.1 Tecnología JMX

Las Extensiones de Java para Gestión (JMX, Java Management Extensions) [Sun99b, Sun99e], previamente conocida como la Interfaz de Programa de Aplicación de Gestión Java (JMAPI, Java Management API), son un conjunto de especificaciones desarrolladas por Sun Microsystems junto con compañías líder en el sector de gestión, siguiendo el Proceso de la Comunidad Java (JCP, Java Community Process) para la definición de extensiones Java.

JMX define una arquitectura de gestión, un conjunto de APIs y servicios de gestión para proporcionar a los desarrolladores de aplicaciones basadas en el lenguaje de programación Java los medios necesarios para la creación de agentes y gestores en Java permitiendo la integración con soluciones de gestión existentes.

La especificación divide las extensiones para gestión en tres niveles diferentes:

- Nivel de Instrumentación.
- Nivel de Agente.
- Nivel de Gestor.

Adicionalmente incluye un conjunto de APIs para protocolos de gestión estándares existentes en la parte referenciada como:

- APIs Adicionales para Protocolos de Gestión.

En la figura 3.5 se muestra la relación entre los distintos niveles y los componentes de cada nivel que serán brevemente introducidos en los apartados siguientes.

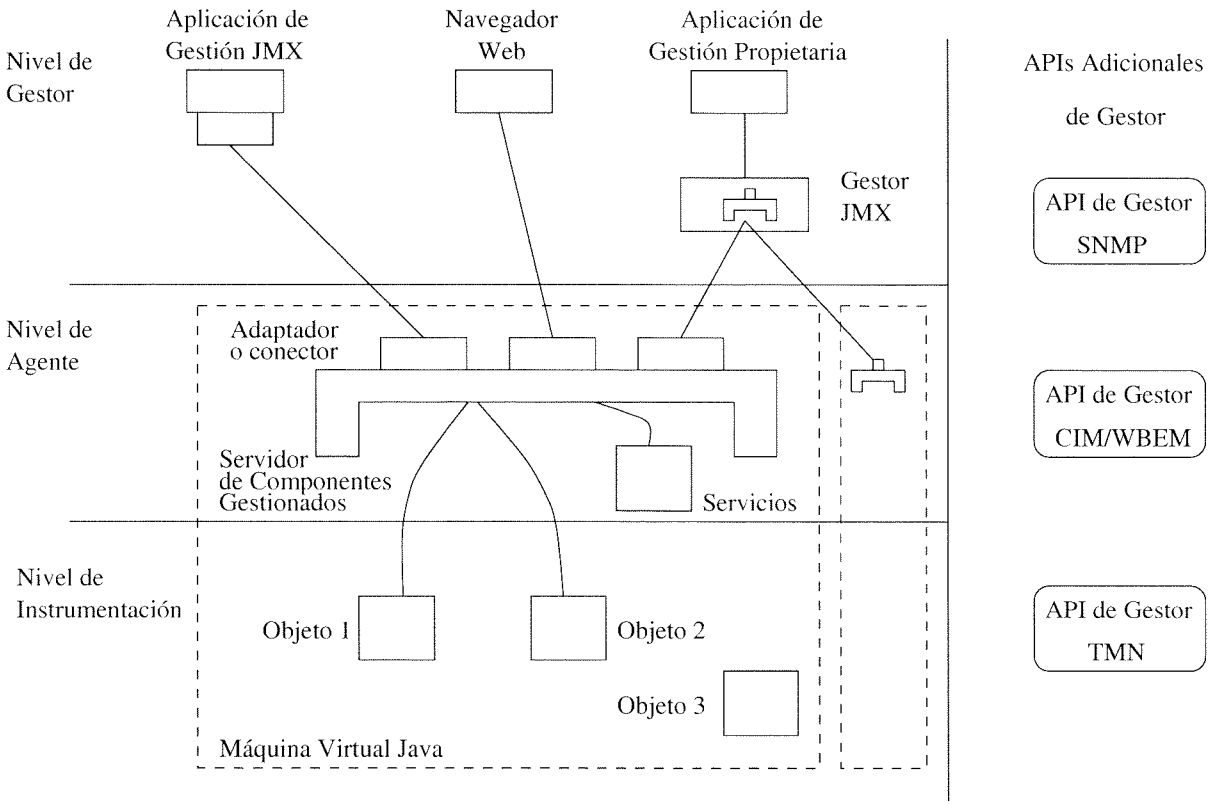


Figura 3.5: Componentes de JMX

Nivel de Instrumentación

El nivel de instrumentación proporciona una especificación para la implementación de recursos gestionables JMX. Un recurso gestionable JMX puede ser una aplicación, una implementación de un servicio, un dispositivo, un usuario, etc. Este recurso está implementado en Java, o al menos ofrece una envoltura Java. La instrumentación de un recurso permite que este recurso sea gestionado por medio del nivel de agente.

Los elementos de este nivel son:

- Componentes Gestionados (MBeans, Managed Beans): Es un objeto Java que implementa interfaces específicas y que sigue el modelo de componentes JavaBeans [Zuk98]. Se definen 4 tipos diferentes:
  - Componente Gestionado Estándard: La interfaz de gestión es descrita por los nombres de sus métodos. Es el más simple de diseñar y de implementar.
  - Componente Gestionado Dinámico: Para obtener mayor flexibilidad, expone la interfaz de gestión en tiempo de ejecución.

  - Componente Gestionado Abierto: Es un componente gestionado dinámico que utiliza tipos de datos básicos para lograr una gestión más universal y autodescritos para ser más entendible por el usuario.
  - Componente Gestionado Modelo: Es también un componente gestionado dinámico que son completamente configurables y autodescritos en tiempo de ejecución.

- Modelo de Notificación: Basado en el modelo de eventos de Java. En el contexto de un agente, las notificaciones pueden ser emitidas tanto por instancias de componentes gestionados como por el servidor del componente gestionado.

Nivel de Agente

El nivel de agente proporciona una especificación para la implementación de agentes. Haciendo uso de la especificación del nivel de instrumentación se define un agente estándar para gestionar los recursos gestionables JMX.

Los elementos clave en este nivel son el servidor de componentes gestionados y los servicios de gestión, aunque debe contener al menos una adaptador de comunicaciones o conector que permite al gestor acceder a este nivel y por tanto a los componentes gestionados y a los servicios proporcionados:

- Servidor de Componentes Gestionados: Es un registro para objetos en el nivel de instrumentación.
- Servicios de Gestión: Permiten integrar inteligencia en el servidor de componentes gestionados para obtener mayor autonomía y rendimiento.

Nivel de Gestor

El nivel de gestor proporciona una especificación para la implementación de gestores JMX. Las principales funcionalidades que son contempladas en este nivel son las siguientes:

- Proporcionar un interfaz a las aplicaciones de gestión que les permita interactuar con el agente a través de los conectores.
- Distribuir información de gestión desde las plataformas de gestión de alto nivel a los distintos agentes.

- Reunir información de gestión procedente de distintos agentes en vistas lógicas que son relevantes para las operaciones de mercado del usuario final.
- Proporcionar seguridad.

APIs Adicionales para Protocolos de Gestión

Las APIs adicionales para protocolos de gestión proporcionan una especificación para interactuar con otros entornos de gestión. En una primera fase las interfaces soportadas son:

- API para el gestor SNMP [Sun99d]: Es un conjunto de clases Java para simplificar el desarrollo de aplicaciones para la gestión de agentes SNMP.
- APIs para el gestor CIM/WBEM [Sun99c]: Es un conjunto de clases Java para simplificar el desarrollo de aplicaciones para la gestión de objetos CIM/WBEM.

Se tienen previstas para fases posteriores APIs para otros entornos de gestión importantes como por ejemplo TMN.

Las APIs adicionales JMX para protocolos de gestión no definen las funciones de las aplicaciones ni la arquitectura de las plataformas, sólo definen APIs Java estándar para acceder a otras tecnologías de gestión como por ejemplo SNMP.

3.9.2 WBEM

La Gestión Basada en Web (WBEM, Web-Based Management) [Wil99] es una iniciativa de la industria liderada por el DMTF (Distributed Management Task Force). WBEM es un conjunto de tecnologías estándar de gestión e Internet desarrolladas para unificar la gestión de entornos de computación de la empresa. El componente principal de esta tecnología es el Modelo de Información Común (CIM, Common Information Model) [Dis99].

CIM es un modelo de datos común de un esquema no dependiente de la implementación para describir información de gestión en un entorno de red/empresa. CIM está compuesto de:

- Especificación: Define los detalles para la integración con otros modelos de gestión (por ejemplo: MIBs de SNMP)

- Esquema: Proporciona las descripciones del modelo real.

El esquema CIM es una colección de definiciones de clases usadas para representar objetos gestionados que se encuentran en todo entorno de gestión. El esquema CIM se divide en tres niveles diferentes:

- Model Núcleo: Incluye clases de alto nivel, sus propiedades y asociaciones. Este modelo es independiente tanto de la plataforma como del dominio.
- Modelo Común: Incluye una serie de clases específicas del dominio pero independientes de la plataforma. Los dominio son: Sistemas, Redes, Aplicaciones y otros datos relacionados con la gestión. El Modelo Común se deriva del modelo anterior.
- Esquema Extendido: Incluye extensiones específicas de la plataforma tal como Windows o UNIX.

WMI (Windows Management Instrumentation) [BMC99] es la implementación Windows de WBEM. Extiende CIM para representar objetos de gestión en entornos de gestión Windows. Microsoft ha anunciado que incluirá WMI como parte de Windows 2000.

3.9.3 Código Móvil

Los Sistemas de Código Móvil permiten transferir dinámicamente programas de una localización a otra de la red. Mediante estos sistemas se puede transferir un programa a un agente para ejecutarlo posteriormente. Tanto la transferencia del programa como la ejecución puede ser comenzada por el agente donde se ejecuta o por una entidad exterior, como puede ser un gestor u otro agente.

Se identifican tres tipos diferentes de metodologías de código móvil:

- Evaluación Remota (REV, Remote EValuation): Un cliente que necesita un servicio de un servidor, no se limita a enviar el nombre del servicio y los parámetros de entrada, sino que envía además el código que debe ser ejecutado en el servidor. Es decir, que el cliente posee el código que se va a ejecutar en el servidor, mientras que el servidor posee los recursos sobre los que se va a ejecutar el servicio y un entorno adecuado para la ejecución.

- Código Bajo Demanda (COD, Code On Demand): Cuando necesita realizar una tarea dada, se pone en contacto con un servidor de código, descarga el código necesario desde el servidor, enlaza de forma dinámica este código y lo ejecuta. En este caso, el cliente posee los recursos necesarios, mientras que el servidor tiene el código a ser ejecutado.
- Agentes Móviles (MA, Mobile Agent): Un agente móvil es una unidad de ejecución que tiene la habilidad de migrar de forma autónoma a otro host y continuar la ejecución. El agente móvil posee el código pero no los recursos.

El paradigma de código móvil ha sido aplicado en un marco de trabajo de gestión denominado Gestión por Delegación [Gol95]. Tanto IETF como OSI han integrado conceptos de código móvil en sus entornos de gestión respectivos. El resultado ha sido un Internet draft definiendo la MIB Script [Lev97], y un draft del ITU-T definiendo la función de gestión del Secuenciador de Órdenes en la recomendación X.753 [ITU97d].

3.9.4 Agentes Inteligentes

El concepto de Agentes Inteligentes es originario de la comunidad DAI (Distributed Artificial Intelligence). Debido a lo complejo que resulta la definición consensuada de Agente Inteligente, éste se define por las propiedades que debe exhibir [Woo95] que son las siguientes:

- Autonomía: Un Agente Inteligente opera sin la intervención directa de humanos y tiene algún tipo de control sobre sus acciones y estado interno.
- Cooperatividad: Los Agentes Inteligentes cooperan entre sí para lograr sus objetivos a través de algún lenguaje de comunicación.
- Reactividad: Un Agente Inteligente percibe su entorno y responde a cambios que ocurren en él de una forma temporizada.
- Proactividad: Un agente inteligente puede tomar la iniciativa para lograr sus objetivos sin necesidad de eventos externos para reaccionar.

Los Agentes Inteligentes pueden adicionalmente tener propiedades opcionales como por ejemplo la movilidad.

Cuando los Agentes Inteligentes cooperan entre sí, se necesita un lenguaje estandarizado para la comunicación entre éstos. El lenguaje KQML (Knowledge Query and Manipulation Language) [Fin94] es el que ha encontrado mayor éxito en la comunidad de gestión de sistemas.

Capítulo 4

Estado del Arte de MMS

En este capítulo se va a presentar un visión general del protocolo MMS. Se van a tratar los beneficios que conlleva la utilización de este protocolo, así como el modelo y los objetos que lo componen. El modelo de objetos de este protocolo se ha tomado como base para la integración de los dispositivos de fabricación en los sistemas basados en CORBA. Se consigue de esta forma un acceso homogéneo a todos los dispositivos que conforman el sistema distribuido, independientemente de su naturaleza física. La última parte de este capítulo se dedica a comentar distintas formas de implementación de MMS.

4.1 Protocolos de Comunicación para Fabricación

En los sistemas distribuidos heterogéneos, tales como los sistemas de fabricación, donde cada dispositivo tiene diferentes características, llevan a cabo diferentes tareas, y pertenecen a distintos fabricantes, aparece la necesidad de interconectar todos los dispositivos que componen el sistema para lograr la integración de cada uno de ellos en el sistema completo.

Se pueden utilizar dos aproximaciones diferentes para permitir comunicar datos entre estos dispositivos [ESP95]:

- Usar herramientas y lenguajes proporcionados por el fabricante de cada dispositivo.
- Usar un lenguaje común para todos los dispositivos. Este lenguaje común debe ser independiente de las características de bajo nivel del dispositivo.

Las desventajas de la primera aproximación, se pueden resumir en las siguientes:

- Necesidad de personal especializado que debe aprender las características particulares de cada dispositivo.
- La aplicación es dependiente del dispositivo. Si se reemplaza un dispositivo por otro con características similares pero de diferentes fabricantes, o es un modelo diferente, la aplicación tendrá que ser modificada.
- Es difícil depurar y mantener aplicaciones para estos sistemas.

La segunda aproximación no tiene estas desventajas. El desarrollador de aplicaciones para estos sistemas sólo necesitará aprender un lenguaje. Además, la aplicación será independiente del dispositivo y por lo tanto más fácil de depurar y mantener.

En este segundo enfoque puede ser incluido el protocolo de Especificación de Mensajes de Fabricación (MMS, Manufacturing Message Specification).

4.2 MMS: Estándar Internacional

MMS es un lenguaje de comunicación que ayuda a la interconexión de dispositivos en entornos heterogéneos. Es un estándar ISO (International Organization for Standadization), ISO 9506, que es desarrollado y mantenido por Technical Committee Number 184 (TC184), Industrial Automation.

Es un protocolo del nivel de aplicación que ofrece servicios para el control y monitorización de dispositivos industriales. Ofrece un amplio rango de servicios para leer y escribir variables, cargar programas y datos, para controlar de forma remota la ejecución de programas, para gestionar eventos, etc.

Se definen en el protocolo:

- Un conjunto de objetos, que representan recursos del dispositivo.
- Un conjunto de servicios, que pueden ser usados para manipular de forma remota estos objetos.
- Un conjunto de mensajes,Unidades de Datos del Protocolo(PDUs, Protocol Data Units), que transportan las peticiones y las respuestas asociadas a estos servicios.

- Un conjunto de autómatas, indicando los servicios que pueden ser llamados en cada estado del dispositivo.

Las distintas partes que componen el estándar están esquematizadas en la figura 4.1. La definición de MMS se compone de una parte central conocida como el núcleo de MMS y un conjunto de anexos. En el núcleo se encuentran la definición de servicio [ISO90a] y la especificación del protocolo [ISO90b]. En la definición de servicio se especifican los objetos que componen el estándar y los servicios que se utilizan para manejar estos objetos. En la especificación del protocolo se utiliza ASN.1 para especificar el formato de los mensajes correspondientes a las peticiones y respuestas de estos servicios. En los distintos anexos se definen subconjuntos de servicios y objetos MMS aplicables a familias de dispositivos industriales, tales como robots o controladores numéricos. Este trabajo se ha centrado en las dos primeras especificaciones, que corresponden a la definición de servicio y a la especificación del protocolo.

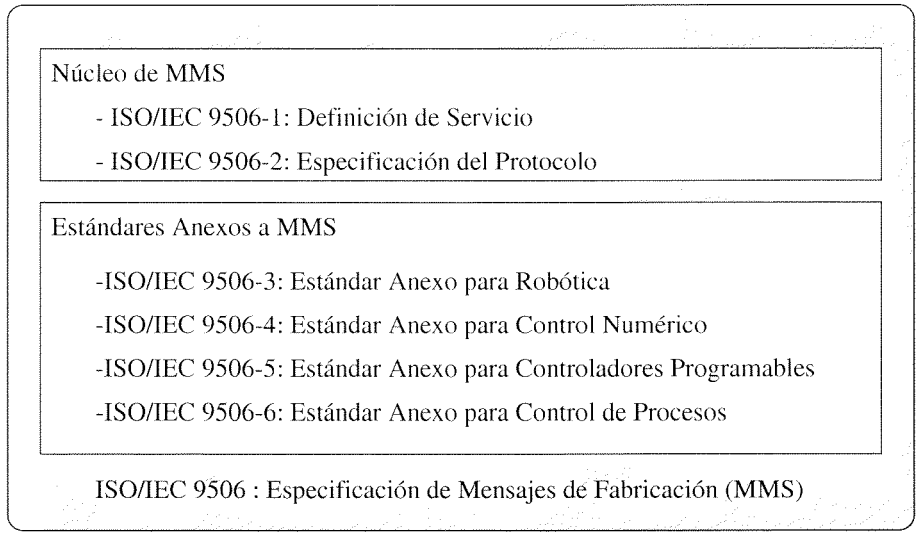


Figura 4.1: Estándar MMS.

4.3 Beneficios de MMS

Los beneficios de la utilización del protocolo MMS en sistemas de fabricación se pueden resumir en los siguientes:

- Reducción de costos de construcción y uso de sistemas automatizados.
- Interoperabilidad: es la capacidad de dos o más aplicaciones de red para intercambiar información de datos de procesos y control de supervisión entre ellos sin que el usuario de la aplicación tenga que crear el entorno de comunicación.

- Independencia de:
  - El desarrollador de la aplicación. Otros esquemas son normalmente específicos de la aplicación o del dispositivo. MMS está definido por estándares internacionales independientes en el que han participado muchos expertos y vendedores de la industria.
  - La conectividad de la red. MMS es el interfaz a la red para las aplicaciones, aislando a la aplicación de la mayoría de aspectos no-MMS de la red y de cómo la red transfiere mensajes desde un nodo a otro.
  - La función realizada. MMS proporciona un entorno de comunicación común independiente de la función realizada. Una aplicación de control de inventario accede a datos de producción en un dispositivo de control, de la misma manera que un sistema de gestión de energía lee los datos de consumo de energía del mismo dispositivo.
- Acceso a datos: es la capacidad de las aplicaciones de red de obtener la información requerida por la aplicación. MMS es lo suficientemente riguroso como para minimizar las diferencias entre aplicaciones que realizan funciones similares, pero es lo suficientemente genérico como para soportar muchos tipos diferentes de aplicaciones y dispositivos.

4.4 Modelo Cliente/Servidor

MMS está basado en el modelo cliente/servidor. El esquema de funcionamiento se muestra en la figura 4.2 y los componentes que se pueden identificar son los siguientes:

- Proveedor del servicio: parte del sistema de comunicación que proporciona al usuario los servicios MMS.
- Servidor MMS: aplicación que mapea a objetos MMS los recursos físicos que pueden ser accedidos a través de la red.
- Cliente MMS: aplicación que accede a los objetos MMS del servidor por medio de la red haciendo uso de los recursos físicos proporcionados por este servidor MMS.

En general los servicios MMS están agrupados en dos categorías:

- Confirmados: Requieren confirmación. El servicio confirmado puede ser requerido tanto por el cliente como por el servidor.Los pasos son los siguientes:

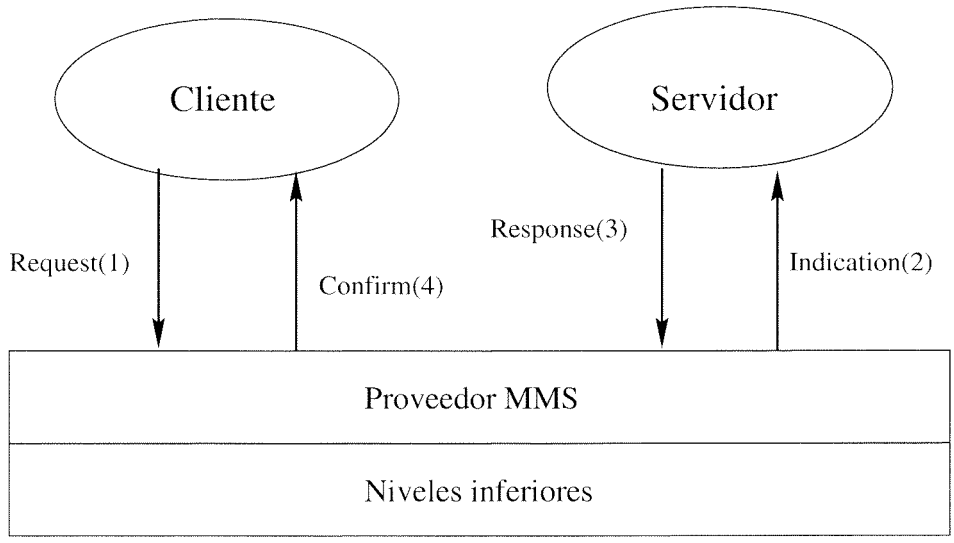


Figura 4.2: Modelo Cliente/Servidor.

- El cliente pide un servicio al proveedor MMS local.
- El proveedor MMS crea la PDU MMS (MMS-Request-PDU) apropiada y pide a los niveles inferiores la transmisión de la PDU.
- El proveedor MMS remoto recibe la petición y crea una indicación de servicio que es enviado al servidor.
- El servidor ejecuta el servicio MMS requerido y envía el resultado a su proveedor MMS local.
- El proveedor MMS del servidor crea una respuesta (MMS-Response-PDU) y se lo envía a los niveles bajos de comunicación, para la transmisión a la estación cliente.
- El proveedor MMS del cliente recibe la respuesta y envía el resultado a su cliente local usando el servicio de confirmación.

- No confirmados: No requieren confirmación. Este tipo de servicio sólo puede ser invocado por un servidor para informar de un evento significante al cliente. Con estos servicios es posible evitar operaciones que consumen tiempo tal como leer periódicamente valores de variables a través de la red. Hay 3 servicios no confirmados:

  - UnsolicitedStatus,
  - InformationReport y
  - EventNotification.

4.4.1 Asociaciones

Una asociación puede ser vista como una conexión a nivel de aplicación que enlaza un cliente a un servidor. En una asociación la comunicación es bidireccional.

Esta asociación se establece mediante el servicio Initiate. En este paso se puede llevar a cabo la negociación de ciertos parámetros, como por ejemplo:

- Número de servicios pendientes.
- Máximo nivel de anidamiento en las estructuras de datos.
- Número de versión.

En el inicio cada usuario MMS indica a su par los servicios que ellos soportan. La asociación se considera establecida cuando hay una respuesta positiva al servicio Initiate.

Además de este servicio de inicio de asociación, MMS cuenta con otros servicios para el control de la asociación. Estos servicios son los siguientes:

- Terminación ordenada: Servicio *Conclude*.
- Terminación abrupta: Servicio *Abort*.
- Cancelación de servicio: Servicio *Cancel*.
- Notificación de servicio no soportado: Servicio *Reject*.

4.5 Modelo de Objetos

Aunque MMS no soporta todas las características de la programación orientada a objetos, divide el sistema en objetos y presenta una interfaz bien definida para cada uno de estos objetos. Los conceptos clave en el modelo de objetos de MMS son:

- Objeto: Representación de una entidad del mundo real.
- Atributo: Información concerniente al objeto.

- Operación: Servicio proporcionado por el objeto.
- Clase: Colección de objetos.
- Instancia: Miembro de una clase.

Los objetos son independientes del dispositivo y de la aplicación. El usuario accede a objetos MMS, no a los objetos reales.

4.6 Definición de Clases

Se usan plantillas para definir las propiedades de una clase de objetos. En una plantilla se incluye la definición de los atributos de la clase. No se incluyen las operaciones de la clase. Cada instancia de una clase de objetos tiene el mismo conjunto de atributos, pero cada una tiene su propio conjunto de valores. Estos valores pueden ser modificados por servicios MMS. Existen distintos tipos de atributos:

- El atributo clave permite identificar una instancia de una clase entre todas las instancias de esa misma clase. Cada clase solo tiene un atributo clave.
- Los atributos condicionales sólo existen si se cumplen ciertas condiciones.
- Los atributos referencias a instancias de otras clases son enlaces a otros objetos y reflejan relaciones entre objetos.

Normalmente los atributos de los objetos pueden ser accedidos a través de servicios MMS, pero puede haber atributos que no son visibles a clientes MMS, y pueden ayudar a considerar aspectos generales de la implementación.

Por ejemplo, la plantilla asociada a la clase Invocación de Programa es la siguiente:

```
Object: Program Invocation
        Key Attribute:  Program Invocation Name
        Attribute:      State (IDLE, STARTING, RUNNING, STOPPING,
                               STOPPED, RESUMING, RESETTING,
                               UNRUNNABLE)
        Attribute:      List Of Domain Reference
        Attribute:      MMS Deletable (TRUE, FALSE)
```

```
Attribute:      Reusable (TRUE, FALSE)
Attribute:      Monitor (TRUE, FALSE)
Constraint      Monitor = TRUE
                Attribute:          Event Condition Reference
                Attribute:          Event Action Reference
                Attribute:          Event Enrolment Reference
Attribute:      Execution Argument
Attribute:      Additional Detail
```

El atributo clave es el nombre del objeto (Program Invocation Name). El nombre suele ser el atributo clave en todas las clases. El nombre de un objeto está enlazado a un espacio de nombre o ámbito. Se definen 3 ámbitos (scope) diferentes: ámbito del Dispositivo de Fabricación Virtual (VMD), ámbito de Dominio (Domain) y ámbito de la Asociación de la Aplicación (Aplication Association).

Los atributos Event Condition Reference, Event Action Reference y Event Enrolment Reference son atributos condicionales y están presentes sólo si el atributo Monitor es TRUE. Son además atributos referencias , es decir, son enlaces a objetos, estos son creados e inicializados automáticamente por el servidor cuando el cliente pide una ejecución monitorizada.

4.7 Objetos MMS

En la especificación de MMS se definen un conjunto de objetos. La definición se realiza mediante las plantillas introducidas en el apartado anterior, donde se definen los atributos. Se definen también un conjunto de servicios para cada unos de estos objetos. Los distintos objetos especificados en el estándar son:

- Dispositivo de Fabricación Virtual (VMD, Virtual Manufacturing Device): Es el objeto más importante de todos, todos los demás son objetos subordinados a éste.
- Dominio: Es un objeto que representa algún tipo de recurso en el VMD. Se utiliza por ejemplo para contener código de programa o datos.
- Invocación de Programa: Representa la ejecución de un programa en el VMD. Permite que un cliente controle el inicio y terminación del programa.
- Variable: Una variable representa a una variable real del VMD.

- Semáforo: Objeto que sirve para controlar el acceso a otros recursos y objetos en el VMD.
- Operador: Se utiliza para la comunicación con el operador, para mostrar una salida en una pantalla y obtener datos de teclado.
- Eventos: El modelo de eventos de MMS permite notificar al cliente acerca de una ocurrencia en el VMD, así como la ejecución de una acción asociada a esta ocurrencia.
- Diario: Representa el registro de datos (o log) basado en el tiempo. Cada entrada en el diario puede contener el estado de un evento, el valor de una variable, o una cadena de caracteres (anotación) que el VMD o el cliente introduce en el diario.
- Fichero: Objeto que representa un fichero en el VMD, permite transferir, renombrar y borrar ficheros. Se proporciona un servicio de directorio para obtener una lista de ficheros disponibles.

A continuación se exponen las clases de objetos más importantes, en los que se ha centrado este trabajo, que son el dispositivo de fabricación virtual, los dominios, las invocaciones de programas, las variables y los eventos.

4.7.1 Dispositivo de Fabricación Virtual

El dispositivo de fabricación virtual (VMD, Virtual Manufacturing Device) es el objeto esencial en MMS. Todos los demás objetos están subordinados al VMD. Es la vista estándar que un cliente tiene de un servidor. Se considera una abstracción del dispositivo real (RMD, Real Manufacturing Device) ocultando los detalles de implementación y las diferencias entre dispositivos de distintos proveedores. Un VMD puede ser reemplazado por otro siempre que exhiban los mismos atributos, servicios y comportamientos. El sistema distribuido de fabricación se estructura como un conjunto de VMD's.

El VMD actúa como un servidor, llevando a cabo un conjunto de servicios que son descritos en el protocolo MMS. Los clientes son las entidades que requieren los servicios proporcionados por el VMD.

Una implementación particular de un servidor MMS debe proporcionar el mapeado entre el modelo del VMD, que es una abstracción, y la funcionalidad del dispositivo de fabricación real.

Los atributos de este objeto pueden ser clasificados en 3 grupos:

- Atributos descriptivos: *Vendor name, model name y revision.*

  Son especificados por el vendedor y contienen información de identificación. Esta información puede ser usada por la aplicación cliente para identificar al dispositivo remoto. El servicio Identify es usada para requerir esta información.

- Atributos de estado: *Logical status y physical status.*

  El estado lógico refleja el nivel real de funcionalidad disponible a través de MMS, informan acerca de los servicios que están soportados (todos, todos los que no causan ningún cambio de estado, todos excepto los de gestión de invocación de programa, etc...). El estado físico refleja el estado operacional real del hardware del dispositivo real (no se han detectado deficiencias, al menos una malfunción del hardware ha sido detectada, no se puede hacer nada útil, proceso de reparación es necesario).

- Atributos listados de objetos subordinados: *list of capabilities, domains, PI,....*

  Son atributos que contienen la lista de los objetos que se encuentran en el VMD. Una capacidad es un recurso definido localmente que puede ser identificado por una cadena de caracteres.

Los servicios del Objeto VMD son los siguientes:

- *Status*: Este servicio permite al cliente obtener el estado del VMD tanto lógico como físico.
- *UnsolicitedStatus*: Es un servicio no confirmado que permite al cliente recibir un mensaje no solicitado acerca del estado del VMD.
- GetNameList: Con este servicio el cliente puede obtener la lista de varios objetos definidos (PI, domains, variables).
- Identify: Permite obtener los atributos descriptivos del VMD que identifican a la entidad de aplicación MMS.
- Rename: Permite cambiar el nombre de un objeto del VMD.
- GetCapabilityList: Este servicio se utiliza para obtener las listas de las capacidades del VMD.

4.7.2 Dominio

El objeto dominio se utiliza principalmente para el control de ejecución de programas en el VMD. Es un objeto que representa un recurso en el dispositivo. Se pueden

representar como dominios por ejemplo la memoria en la que se almacena un programa, las herramientas disponibles de un centro de fabricación flexible o un brazo de robot en un sistema de robótica.

En general puede contener cualquier tipo de dato que pueda ser descargado (uploaded) desde el VMD a un dispositivo externo o descargado (downloaded) desde un dispositivo externo al VMD. En la práctica, el dominio contiene objetos de otras clases, por ejemplo, variables, instrucciones de un programa, etc. El contenido de un dominio puede ser visible o invisible al cliente. Hay tres formas de crear un dominio, durante el procedimiento de descarga, predefinidos por el sistema o por medios locales. Los servicios del Objeto Domain son los siguientes:

- *InitiateDownloadSequence*: Permite al cliente MMS comenzar una secuencia de descarga para cargar un dominio en el servidor.
- *DownloadSegment*: Permite al servidor MMS obtener un segmento de la información a ser descargada en el VMD.
- *TerminateDownloadSequence*: Permite al servidor MMS terminar la secuencia de descarga.
- *InitiateUploadSequence*: Permite al cliente MMS comenzar el proceso de carga del contenido de un dominio específico.
- *UploadSegment*: Permite al cliente obtener un segmento de la información que se quiere descargar desde el servidor MMS.
- *TerminateUploadSequence*: Este servicio permite al cliente MMS terminar la secuencia de carga.
- *RequestDomainDownload*: Es usado por el servidor para requerir que el cliente inicie una secuencia de descarga.
- *RequestDomainUpload*: Es usado por el servidor para requerir que el cliente inicie una secuencia de carga.
- *LoadDomainContent*: Permite a un cliente pedir que el servidor realice la acción adecuada para cargar un dominio de un fichero o de una tercera parte.
- *StoreDomainContent*: Permite a un cliente pedir que el servidor transfiera los datos del dominio a un fichero o a una tercera parte.
- *DeleteDomain*: Este servicio se utiliza para borrar el dominio.
- *GetDomainAttributes*: Permite al cliente obtener los atributos de un dominio.

En la figura 4.3 se muestran las distintas peticiones de servicio que se llevan a cabo para realizar la carga de un dominio en el cliente desde el servidor.

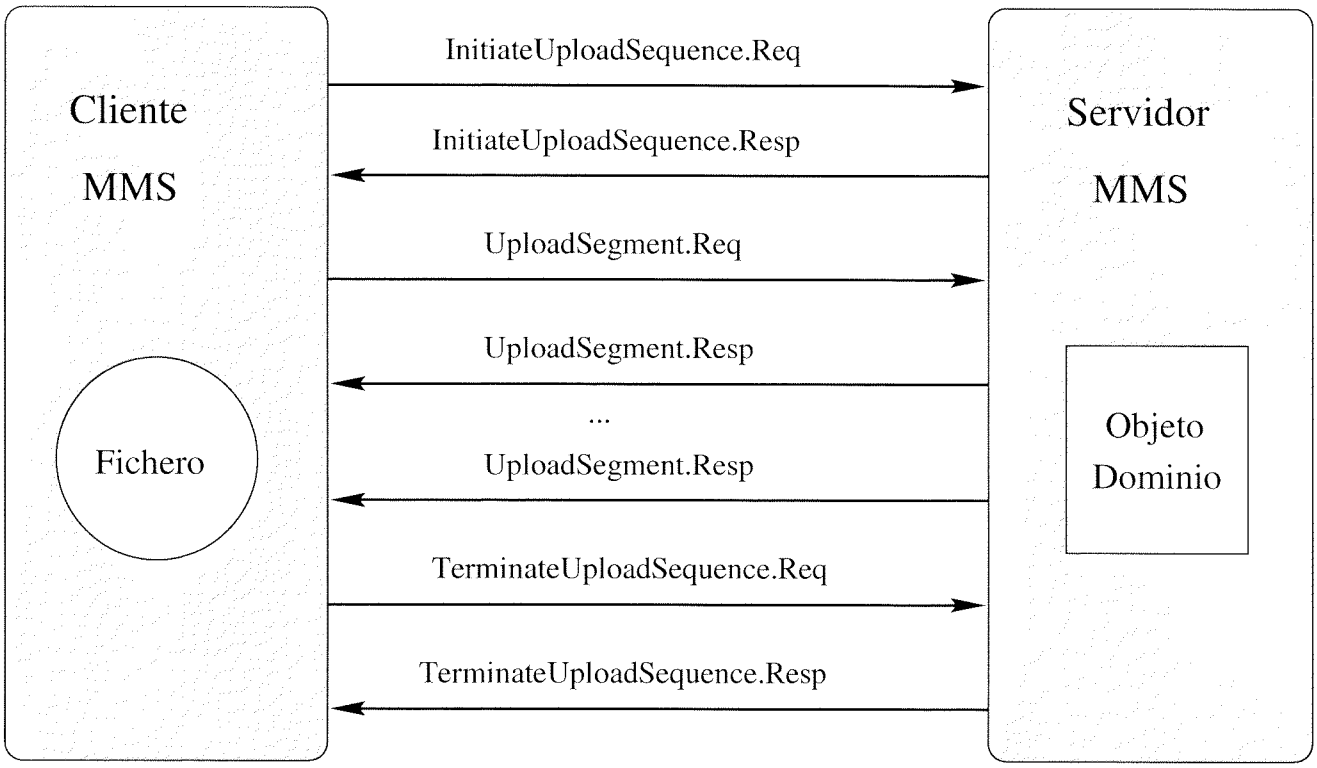


Figura 4.3: Carga de un Dominio.

4.7.3 Invocación de Programa

Describe el comportamiento visible externamente de una tarea o hilo de ejecución. Está compuesto por uno o más dominios que contienen el código y los datos del programa. El dominio que contiene el código puede ser compartido por varios objetos invocación de programa. El objeto invocación de programa puede ser predefinido, creado por medios locales o mediante servicios MMS. Los servicios de este objeto son:

- *CreateProgramInvocation*: Permite al cliente enlazar dominios a un objeto invocación de programa especificado del servidor.
- *DeleteProgramInvocation*: Es utilizado por el cliente para borrar un objeto invocación de programa especificado en el servidor.
- *Start*: Permite al cliente comenzar la ejecución del programa representado por un objeto invocación de programa.
- *Stop*: Se utiliza para parar la ejecución de un programa asociado a un objeto invocación de programa del servidor.
- *Resume*: Permite al cliente continuar con la ejecución del programa parado mediante el servicio stop.

- *Reset*: Permite al cliente restaurar el estado del objeto invocación de programa a IDLE.

- *Kill*: Permite al cliente poner al objeto invocación de programa en el estado UNRUNNABLE.

- *GetProgramInvocationAttributes*: Este servicio sirve para obtener los atributos asociados con el objeto invocación de programa.

4.7.4 Variable

Los datos en los dispositivos son modelados como variables. Las variables son mapeadas a variables MMS. Una variable MMS describe la vista externa de una variable real. En MMS se definen las clases de objetos:

- Variable no Nombrada (Unnamed Variable): Una variable no nombrada es un objeto MMS que representa una variable cercana a la arquitectura física del sistema real. Modela aspectos dependientes del dispositivo. Se describe el acceso a la variable real usando una dirección específica del VMD. Requiere una dirección conocida y fija para la variable. Por ser dependiente del dispositivo representado por el VMD no es aconsejable su uso.

- Variable Nombrada (Named Variable): Una variable nombrada es un objeto que modela la vista de la aplicación de una variable real en el VMD. Describe el acceso a la variable real usando un nombre determinado por el proceso de aplicación. En la práctica son predefinidas o su creación es realizada por medios locales.

- Tipo Nombrado (Named Type): Un tipo nombrado Proporciona la asignación de un nombre a una descripción de tipo MMS.

- Acceso Extendido (Scattered Access): Estructura construida compuesta de variables MMS independientes. Cada componente de esta estructura será un objeto de tipo Named Variable, Unnamed Variable o Scattered Access. El acceso es similar al acceso de una variable simple.

- Lista de Variables Nombradas (Named Variable List): Una lista de variables nombradas es una estructura construida compuesta de variables MMS independientes. Cada componente de esta estructura será un objeto de tipo Named Variable, Unnamed Variable o Scattered Access. El acceso es similar al acceso independiente de las variables de la lista, no se ve como un acceso a una variable simple que es el caso del objeto Scattered Access.

Los servicios para el tratamiento de variables MMS son básicamente para la definición y borrado de los objetos expuestos anteriormente, así como para la obtención de atributos y finalmente las operaciones más importantes que son para la lectura y escritura de datos. Los servicios MMS separados por categorías son los siguientes:

- Servicios de definición de objetos:*DefineNamedVariable, DefineScatteredAccess, DefineNamedVariableList y DefineNamedType.*
- Servicios de borrado de objetos: *DeleteVariableAccess, DeleteNamedVariableList y DeleteNamedType.*
- Servicios de obtención de atributos: *GetVariableAccessAttributes, GetScatteredAccessAttributes, GetNamedVariableListAttributes, GetNamedTypeAttributes.*
- Servicios de lectura/escritura: *Read, Write e InformationReport.*

4.7.5 Evento

El modelo de Gestión de Eventos de MMS proporciona el marco para acceder y gestionar los aspectos de comunicación de eventos (cambios en los valores de las variables). Se definen tres objetos en este modelo, Condición de Evento (Event Condition) que representa el estado de un evento, Registro de Evento (Event Enrollment) que identifica a quién notificar acerca de la ocurrencia de un evento y Acción de Evento (Event Action) que permite especificar una acción que debe ser realizada cuando se produce el evento.

- Condición de Evento

  Representa el estado actual de una condición en el VMD. No define la acción a realizar. En MMS no se define el mapeado del valor de la variable y el estado de la variable condición. Existen dos clases de objetos condición de evento:

  - Disparado por Red (Network Triggered): El cliente MMS dispara el evento mediante el servicio TriggerEvent.
  - Monitorizado (Monitored): Tiene una variable booleana asociada.

- Acción de Evento

  Representa la acción que el VMD realizará cuando el estado de la condición de evento varíe. Es opcional. Es definida como una petición de servicio confirmado (Start, Stop, Read). La respuesta es incluida en la petición de servicio EventNotification que es enviado al cliente. No pueden ser utilizados los servicios no confirmados ni los servicios que deben ser utilizados en conjunción con otros servicios.

- Event Enrollment

  Enlaza todos los elementos del modelo de gestión de eventos MMS. Representa una petición de un cliente MMS de ser notificado acerca de los cambios en el estado de una condición de evento. Se hace referencia a la condición de evento, a la acción de evento (opcional) y al cliente al que hay que enviar la notificación.

Los servicios definidos en el modelo de eventos se pueden dividir en las 4 siguientes categorías:

- Creación y borrado de objetos: *DefineEventCondition, DeleteEventCondition, DefineEventAction, DeleteEventAction, DefineEventEnrollment, DeleteEventEnrollment.*

- Notificación de eventos: *TriggerEvent, EventNotification, AcknowledgeEventNotification.*

- Adquisición de atributos y de estados: *GetEventConditionAttributes, GetEventActionAttributes, GetEventEnrollmentAttributes, ReportEventConditionStatus, ReportEventActionStatus, ReportEventEnrollmentStatus.*

- Obtención de sumarios de objetos de eventos: *GetAlarmSummary, GetAlarmEnrollmentSummary.*

4.8 MMS/MAP

En un principio, MMS surge como el nivel de aplicación del protocolo MAP (Manufacturing Automation Protocol) [Pim90].

MAP es una pila de protocolos de comunicación estándar elegidos para permitir la interoperabilidad entre dispositivos industriales heterogéneos (PLCs, controladores numéricos, computadores industriales, sensores y actuadores inteligentes ...). MAP se ajusta a la arquitectura abierta OSI propuesta por ISO.

A pesar de la importancia que llegó a adquirir MAP en la industria y en diversos proyectos de investigación, no ha tenido una completa aceptación debido principalmente a la complejidad de la pila de protocolos de ISO.

4.9 MMS/ISODE

Debido a la importancia adquirida por TCP/IP y a que se ha convertido en un estándar de facto totalmente asumido por la industria, se plantea la necesidad de la implementación del protocolo MMS sobre TCP/IP.

La aproximación más directa a esta implementación es la especificada en la RFC2126: "ISO Transport Service on top of TCP" [RFC97]. Este documento define un servicio de transporte con los servicios e interfaces ofrecidas por el Servicio de Transporte ISO pero que implementa el Protocolo de Transporte ISO sobre TCP/IP en vez de hacerlo sobre el Servicio de Red de ISO.

Este método ha sido desarrollado en el Proyecto Isode y ha sido tomado como base para la implementación de MMS sobre TCP/IP.

4.10 MMS/RPC

En [Gre95] se aborda también la implementación del protocolo MMS sobre TCP/IP. Esta solución opta por utilizar las RPC's de SUN Microsystem para realizar las interacciones entre el cliente y el servidor de MMS. Desde el punto de vista del cliente, la ejecución de un servicio MMS en la parte del servidor es vista como la llamada a un procedimiento remoto.

El intercambio de mensajes entre entidades que cooperan entre sí es reemplazado por llamadas a procedimientos remotos. Las RPC's son una herramienta muy utilizada para el desarrollo de aplicaciones distribuidas. Sin embargo, las RPC's están siendo sustituidas por las plataformas de objetos distribuidos que proporcionan una forma más adecuada para el desarrollo de aplicaciones distribuidas orientadas a objetos. Aunque de hecho, a veces estas plataformas están basadas en RPC's.

4.11 COOL-MMS

Por último, en [Guy97] se presenta la adaptación de MMS a un entorno orientado a objetos distribuidos.

El prototipo ha sido construido sobre el micro-kernel CHORUS usando su ORB que se ajusta a la especificación CORBA 2.0 llamado COOL.

La utilización de CORBA da más flexibilidad a las aplicaciones distribuidas ya que va a permitir comunicar procesos ejecutándose en plataformas hardware y software diferentes (diferente máquina, diferente sistema operativo y diferente lenguaje de programación).

En [Lau98] se presenta la misma solución anterior pero implementada en Java con la consiguiente mejora que esto supone en portabilidad.

Estas soluciones sin embargo no incluyen el modelo de eventos de MMS que es esencial para las notificaciones de ocurrencias de eventos en un sistema de fabricación.

Por otra parte, la traducción a IDL no sigue la especificación del JIDM. En cada caso se ha realizado la traducción que se ha creído conveniente.

Este trabajo de tesis, ha tomado como base esta aproximación para realizar la adaptación de MMS a CORBA teniendo en cuenta los 2 puntos mencionados anteriormente.

Capítulo 5

Modelo de Gestión para Sistemas de Fabricación

La complejidad para desarrollar aplicaciones para monitorizar y gestionar dispositivos en un sistema distribuido y heterogéneo lleva a la necesidad de utilizar un método adecuado para acceder a los distintos dispositivos de una forma independiente a las características físicas y concretas de cada uno de ellos. MMS es un lenguaje adecuado para llevar a cabo la comunicación entre estos dispositivos y entre ellos y la aplicación de gestión, ya que proporciona uniformidad en el acceso. Para facilitar la implementación del protocolo MMS se usa CORBA.

Se propone un modelo para la gestión de sistemas de fabricación usando MMS como base de la interfaz que sirve para comunicar el gestor con los diversos dispositivos del sistema y usar CORBA como la arquitectura que se responsabiliza de la comunicación necesaria para gestionar los objetos distribuidos.

En este capítulo se comentan las características de los sistemas distribuidos de fabricación, se presenta entonces las arquitecturas de gestión basadas en CORBA y se expone el modelo de integración de dispositivos de fabricación en estos sistemas. Se finaliza el capítulo comentando otras alternativas posibles para gestionar estos dispositivos.

5.1 Sistemas Distribuidos de Fabricación

Debido a las ventajas y funcionalidades aportadas por las redes de comunicación, también los sistemas de fabricación han sido influenciados por éstas. Los distintos compo-

nentes del sistema de fabricación pueden acometer una tarea común haciendo uso de la posibilidad de comunicación que le aporta la red.

Cabe destacar la naturaleza heterogénea de estos sistemas distribuidos. Entre sus componentes se pueden mencionar robots, controladores numéricos, autómatas programables, entre otros dispositivos que son propiamente los que llevan a cabo los distintos procesos de fabricación, así como elementos propios para la interconexión entre estos dispositivos.

El esquema general de un sistema de fabricación integrada se puede observar en la figura 5.1.

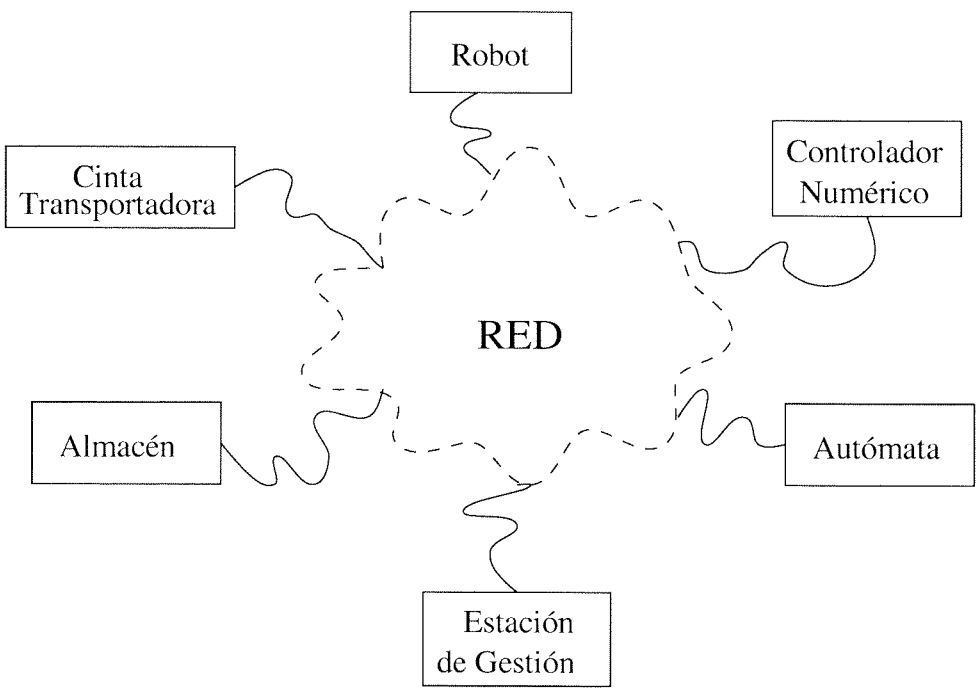


Figura 5.1: Sistema Distribuido de Fabricación.

Una aplicación distribuida de fabricación es una aplicación que hace uso de los dispositivos de fabricación que se encuentran en el sistema necesitando comunicar datos por la red para llevar a cabo la tarea distribuida. Son aplicaciones en las que el tiempo es importante debido a la naturaleza de tiempo real de estas aplicaciones. Un fallo en alguno de los dispositivos debe ser detectado y corregido en el menor tiempo posible.

En este trabajo de tesis se van a tratar estos sistemas desde el punto de vista de la gestión para abordar la necesidad de detectar un fallo en el sistema y la corrección por parte de un operador. La funcionalidad aportada desde este punto de vista no se

limita únicamente al control de fallos, ya que se puede incluir también la monitorización, configuración e inventario de dispositivos.

5.2 Modelo Propuesto

El modelo propuesto en esta tesis pretende avanzar en la integración de los dispositivos de un sistema de fabricación en sistemas de gestión basados en CORBA. Para ello se exponen los distintos aspectos que se han tenido en cuenta:

- Componentes Arquitecturales: Definen los distintos elementos que configuran generalmente el sistema de gestión basado en CORBA.
- Modelo de Integración: Describe el modelo y los elementos que se han de tener en cuenta para lograr la gestión en un entorno de fabricación.
- Interfaz Computacional: Trata de la interfaz basada en MMS que se ha utilizado en los objetos gestionados específicos para fabricación.
- Modelo de Información: Especifica la forma en la que los datos son accedidos en el modelo de gestión que se presenta.

Para la elaboración de este modelo se han utilizado dos tecnologías base, CORBA y MMS, que dan soporte a la gestión del sistema de fabricación. La elección de CORBA se ha realizado teniendo en cuenta que es una tecnología que tiene las siguientes características:

- Permite la integración de componentes desarrollados en distintos lenguajes de programación, permitiendo incluso la interoperabilidad entre distintos sistemas operativos.
- Facilita la comunicación entre los distintos elementos que componen el sistema, ocultando al programador los detalles subyacentes de la comunicación.
- Logra una gran flexibilidad, permitiendo no sólo el acceso a datos sino la ejecución de métodos en los objetos gestionados.
- Se cuenta con un gran número de implementaciones en el mercado ya que ha sido extensamente aceptado por la industria.
- En la actualidad los sistemas de gestión tienden a utilizar CORBA como base para la comunicación.

MMS ha sido elegido como base para la definición de interfaces debido a que:

- Se ajusta bien a la metodología orientada a objetos. Aunque no cuenta con todas las características de la orientación a objetos, sí presenta una visión del dispositivo de fabricación en forma de objetos, con una interfaz adecuada para el acceso a éstos.
- Se utiliza en la industria para la comunicación entre dispositivos de fabricación que componen la aplicación distribuida.
- Es un estándar internacional que como se ha visto en el capítulo de MMS consta de la definición del servicio y de la especificación del protocolo.
- Se pretende utilizar una interfaz uniforme con todos los dispositivos de fabricación, independientemente de su naturaleza. Esta uniformidad se obtiene en MMS.

5.3 Componentes Arquitecturales

Los componentes que forman parte del modelo para la gestión de sistemas pueden ser esquematizados como se muestra en la figura 5.2 y son los siguientes:

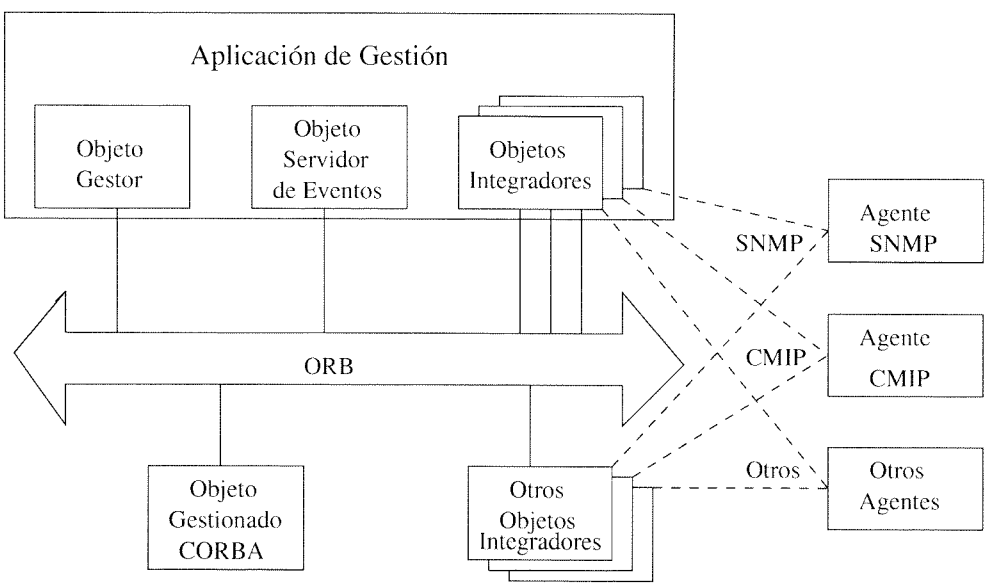


Figura 5.2: Arquitectura de Gestión en sistemas CORBA.

- Objeto Gestor: Objeto que tiene la funcionalidad de gestor, es decir, mediante la utilización de la interfaz del objeto gestionado lleva a cabo labores de gestión.

- Objeto Servidor de Eventos: Objeto que se encarga de la recepción de eventos asíncronos que son comunicados por el objeto gestionado.

- Objeto Gestionado: Objeto que presenta una interfaz que permite al gestor realizar operaciones sobre él. Estas operaciones pueden ser de diversa índole, pueden ser tan simples como la obtención de datos almacenados en el objeto gestionado, u otras más complejas como por ejemplo la ejecución de un programa.

- ORB: Es el agente de peticiones de objetos definido en la arquitetura CORBA que permite la comunicación entre el gestor y los distintos objetos gestionados.

- Objetos Integradores: Son objetos que permiten la integración de dispositivos que son gestionados utilizando otros protocolos de comunicación, tales como SNMP o CMIP.

5.3.1 Objeto Gestor

El objeto gestor es el componente dentro de la aplicación de gestión que realiza las labores de gestión sobre el objeto gestionado. El Objeto Gestor se encarga de:

- Realizar las peticiones de servicio a los objetos gestionados CORBA mediante llamadas a los métodos de la interfaz de estos objetos.

- Obtener los resultados de las peticiones de servicio en los valores de vuelta de los métodos o bien en los parámetros de salida.

- Facilitar a la aplicación de gestión la presentación de los resultados obtenidos para procesamiento por parte del usuario.

Para realizar las llamadas a los métodos del objeto gestionado puede utilizar tanto el método estático como el dinámico:

- Método Estático: En este caso se apoya en el stub o soporte del cliente que debe ser conocido en tiempo de compilación. El soporte permite al gestor realizar las llamadas sin tener que preocuparse de la codificación de los parámetros ni del envío o recepción de éstos.

- Método Dinámico: En este otro caso se apoya en la interfaz de invocación dinámica de CORBA. Mediante este mecanismo, el gestor no tiene que conocer la interfaz del objeto gestionado en tiempo de compilación, ya que puede ser descubierta en tiempo de ejecución.

Cada uno de estos dos métodos tiene sus ventajas e inconvenientes. El método estático es más simple de utilizar pero se necesita información que debe estar disponible en tiempo de compilación. El método dinámico es más complicado de utilizar, pero es muy útil cuando no se cuenta con esta información en tiempo de compilación y hay que obtenerla en tiempo de ejecución. En cada caso se utilizará el que sea oportuno, teniendo en cuenta que en algunos casos particulares no se tiene otra opción más que utilizar el método dinámico.

5.3.2 Objeto Servidor de Eventos

El objeto servidor de eventos es el objeto dentro de la aplicación de gestión que se encarga de procesar las notificaciones de eventos generadas de forma asíncrona por el objeto gestionado o por los objetos integradores. El objeto Servidor de eventos puede seguir dos esquemas diferentes para la recepción de estas notificaciones:

- Presentar una interfaz IDL para que el objeto que genera la notificación realice una llamada a algún método de la interfaz.
- Utilizar el servicio de eventos de CORBA y asumir el rol de consumidor de eventos.

Estos dos esquemas son detallados para el caso de dispositivos de fabricación en el capítulo 7: "Modelos para la Comunicación de Eventos".

5.3.3 Objeto Gestionado

Un objeto gestionado es un objeto CORBA que presenta una interfaz IDL que permite realizar funciones de gestión sobre un dispositivo gestionado. Este objeto es dependiente del dispositivo, pero presenta una interfaz que es independiente del dispositivo. La interfaz constará de métodos para el acceso y modificación de datos, así como métodos más complejos que permitan realizar funciones de valor añadido sobre el dispositivo de una forma más eficiente y cómoda.

5.3.4 Objetos Integradores

Un objeto integrador es un objeto CORBA que proporciona acceso a agentes que utilizan otros protocolos de comunicación (como por ejemplo SNMP o CMIP) pero que no presentan una interfaz IDL a la forma de CORBA.

La interfaz presentada por estos objetos integradores depende del tipo de agente al que se quiere dar acceso. Se puede ver al objeto integrador como una envoltura del agente que oculta los detalles del protocolo concreto utilizado.

Los objetos integradores en sistemas de fabricación se verán en apartados siguientes. Para el caso de otros protocolos, su discusión queda fuera del ámbito de esta tesis.

5.3.5 ORB

El ORB es el agente de peticiones de objetos de CORBA ya comentado en el capítulo 3: "Estado del Arte de CORBA" . El ORB se encarga de:

- Facilitar la llamada a los métodos de los objetos que se encuentran en el bus de objetos.
- Ocultar los detalles de la codificación de parámetros.
- Mantener referencias universales a los distintos objetos.

5.4 Modelo de Integración

El modelo de integración pretende facilitar la gestión, usando CORBA, de dispositivos de fabricación que utilizan el protocolo MMS, así como de facilitar la integración de aquellos que no utilizan directamente MMS pero que pueden presentar una interfaz que se adapta a este protocolo. De esta forma, se contempla en este esquema tanto aquellos dispositivos que inicialmente cuentan con una implementación de MMS como aquellos que no.

El modelo de integración propuesto en este trabajo de tesis se basa en dos formas posibles de realizar la inclusión en el sistema de gestión basado en CORBA. Estas dos formas diferentes son las siguientes:

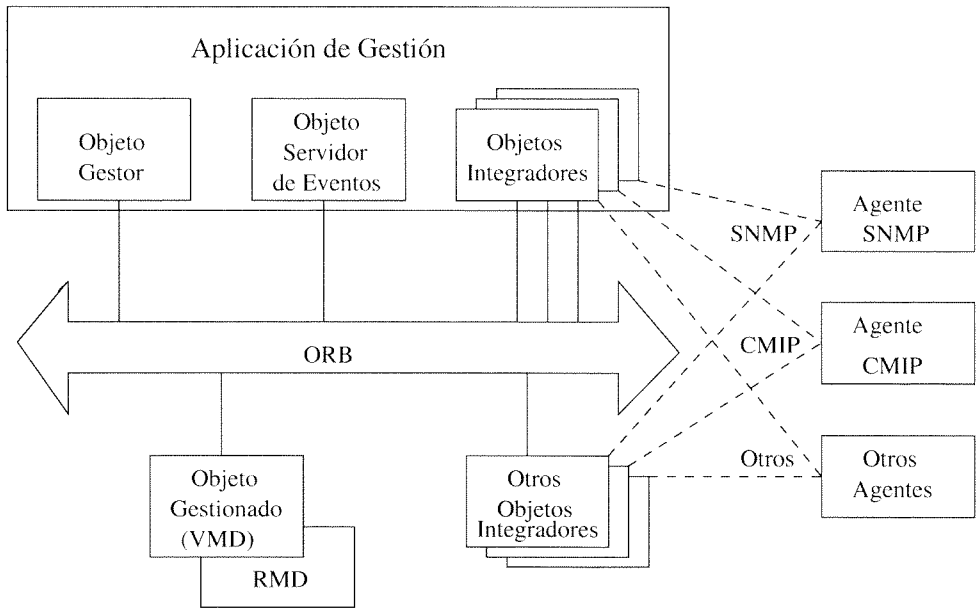


Figura 5.3: Acceso Mediante un Objeto Gestionado CORBA.

- Acceso al dispositivo mediante un objeto gestionado CORBA: El objeto gestionado presenta una interfaz que permite realizar operaciones en él. Esta es la forma que se ha adoptado en el prototipo desarrollado como parte de esta tesis. Este prototipo se comenta en el apéndice C. La figura 5.3 esquematiza esta forma de integración.

- Acceso al dispositivo mediante un objeto integrador: En este caso se pueden considerar dos posibilidades:

  - Objeto integrador en la aplicación de gestión: Es la propia aplicación de gestión la que incluye las facilidades para la gestión de dispositivos de esta naturaleza. La figura 5.4 esquematiza esta forma de integración.
  - Objeto integrador externo a la aplicación de gestión: La aplicación de gestión no cuenta con estas facilidades y es entonces un elemento integrador adicional externo a la aplicación de gestión el que se encarga de presentar la interfaz IDL al dispositivo. La figura 5.5 esquematiza esta forma de integración.

Tanto si se utiliza el objeto gestionado CORBA como si se utiliza el objeto integrador, la interfaz en ambos casos es la misma. Esta interfaz será comentada en el apartado "Interfaz del Objeto Computacional".

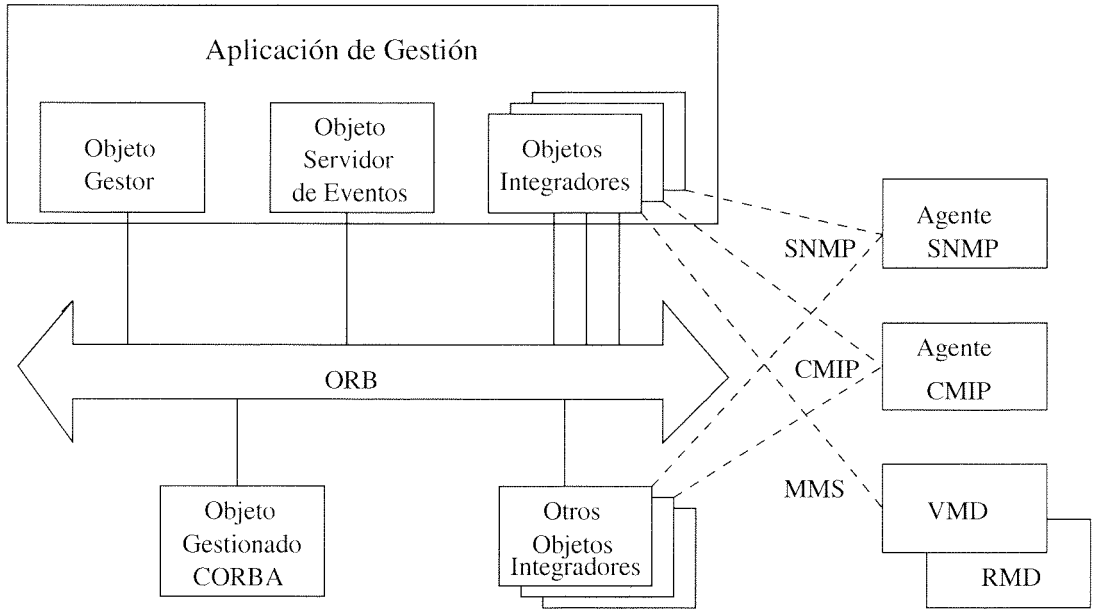


Figura 5.4: Objeto Integrador en la Aplicación de Gestión.

5.5 Interfaz del Objeto Computacional

La interfaz adoptada es una adaptación de los servicios MMS a CORBA. En el capítulo siguiente se expone como se realiza esta adaptación así como la especificación que se ha seguido.

Como se ha comentado previamente en el capítulo referente a CORBA, la interfaz debe ser expresada en el lenguaje de definición de interfaz IDL. En la especificación de la interfaz se expresan las declaraciones de cada uno de los métodos u operaciones que pueden ser ejecutadas en el objeto CORBA a través del ORB de una forma similar a la llamada a un método de un objeto local.

En la definición de la interfaz se incluyen los métodos correspondientes a los servicios MMS, tanto confirmados como no confirmados y tanto los correspondientes al servidor como los correspondientes al cliente. El haber utilizado los servicios proporcionados por MMS da un gran conjunto de posibilidades, ya que, como se ha visto en el capítulo correspondiente a este protocolo, se cuentan con servicios para:

- Lectura y escritura de variables.
- Control de ejecución de programas.

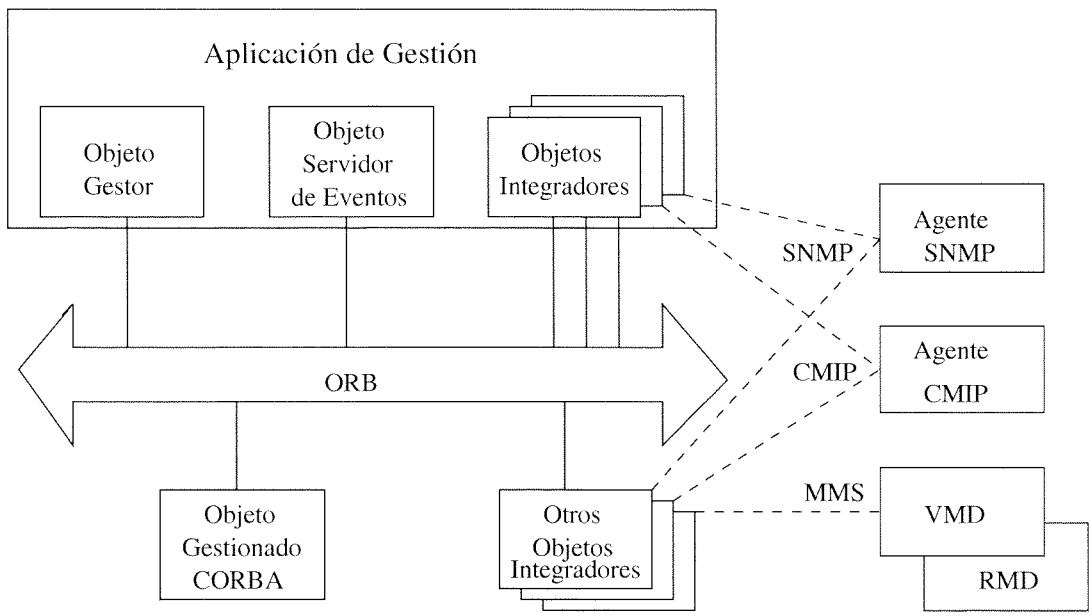


Figura 5.5: Objeto Integrador Externo a la Aplicación de Gestión.

- Carga y descarga de código de programas y datos.
- Comunicación de eventos de una forma asíncrona.

5.6 Modelo de Información

El modelo de información adoptado se ajusta al modelo de información de MMS. Este modelo ha sido extensamente introducido en el capítulo 4: "Estado del Arte de MMS", en el apartado correspondiente al Modelo de Objetos.

Como resumen de lo que ya se ha explicado en este capítulo se puede indicar que:

- El modelo de información especifica un conjunto de objetos. Cada objeto cuenta con sus propios atributos y operaciones.
- El objeto más importante es el VMD que representa a un dispositivo de fabricación real.
- Los demás objetos son objetos subordinados al VMD. Estos objetos son los dominios, las invocaciones de programas, las variables y los eventos entre otros.

- Cada objeto es referido mediante un nombre. Este nombre es único dentro del ámbito de su definición y dentro de la clase de objetos al que pertenece.

5.7 Otras Alternativas

Es este trabajo de tesis se ha contemplado la integración de dispositivos de fabricación en sistemas de gestión basados en CORBA, considerando la importancia que están teniendo actualmente. Otras posibilidades para la gestión de dispositivos de fabricación son las siguientes:

- Gestionar estos dispositivos mediante SNMP: Con esta solución se podrían integrar en sistemas tradicionales de gestión que utilizan este protocolo. Se gana en simplicidad ya que este protocolo es muy simple, pero también se reducen bastante las operaciones de gestión que se pueden realizar. Hay que tener en cuenta que el acceso a un dispositivo MMS mediante SNMP necesita una adaptación previa de estos dos protocolos.

- Gestionar estos dispositivos mediante CMIP: De esta forma se pueden integrar en sistemas tradicionales que utilizan este protocolo. La integración de aquellos dispositivos que utilizan MMS necesitarán una adaptación del modelo MMS al modelo utilizado por CMIP.

- Gestionar estos dispositivos mediante CORBA pero utilizando en cada caso la interfaz que se considere oportuna. En este caso se ganaría en flexibilidad ya que la interfaz no se tendría que ajustar a ningún protocolo establecido sino que cada dispositivo presentaría la interfaz adecuada en cada caso pero se perdería la uniformidad en el acceso a los distintos dispositivos de fabricación mediante el uso de MMS. Esto conllevaría que la aplicación de gestión tendría que tratar con multitud de dispositivos cada uno de ellos con una interfaz diferente.

Capítulo 6

Adaptación de MMS a CORBA

En este capítulo se exponen los principios seguidos para la adaptación del protocolo MMS a CORBA. Esta adaptación se basa en la aproximación COOL-MMS [Guy97] con dos diferencias fundamentales:

- Se ha tenido en cuenta el servicio de eventos de MMS. Este servicio se considera muy importante desde el punto de vista de la gestión y necesita consideraciones adicionales que serán tratadas en este capítulo y el siguiente.
- El esquema de traducción del protocolo MMS a IDL es el especificado por el grupo de trabajo JIDM.

En este capítulo se va a exponer la adaptación del modelo de objetos y el de comunicación de MMS a CORBA así como la adaptación de los servicios y su traducción a IDL.

6.1 Modelo de objetos

El modelo de objetos que se ha seguido para la adaptación de MMS a CORBA se basa en la agregación de los objetos especificados en el estándar MMS (se han tenido en cuenta las variables, dominios, invocación de programas y eventos) en el objeto CORBA mmsServer. Este objeto presenta la interfaz a todos los servicios, tanto a los servicios generales del Dispositivo de Fabricación Virtual como a los servicios de los demás objetos. Se considera entonces que el objeto mmsServer es el encargado de la

creación de los demás objetos que lo componen y delega en ellos los servicios propios de cada uno.

El objeto VMD puede ser accedido a través del bus de objetos de CORBA de una forma transparente mediante el objeto CORBA mencionado anteriormente. Los clientes pueden llamar a los métodos de este objeto de una forma remota.

Por otra parte, el objeto CORBA correspondiente al cliente mms presenta la interfaz a los servicios que son invocados desde el servidor al cliente.

El modelo de objetos del servidor y del cliente expresado mediante la técnica de modelado de objetos OMT puede ser consultada en el apéndice C donde se comenta el prototipo realizado en Java.

6.2 Servicios

Como ya se ha comentado en el capítulo de MMS, los servicios pueden ser confirmados o no confirmados. La traducción para estos dos tipos de servicios es la siguiente:

- Servicios confirmados: Son traducidos a métodos de invocación síncrona. El cliente hace la llamada al método del VMD, se bloquea hasta que no recibe la respuesta, es decir, hasta que no termina la ejecución del método y obtiene los resultados de la llamada.

- Servicios no confirmados: Son traducidos a métodos con el atributo de operación "oneway" que se comenta en "Apéndice A: OMG IDL". Estos servicios son normalmente servicios que permiten al VMD enviar información no solicitada al cliente. Por ejemplo, los servicios de envío de estado no solicitado y notificación de eventos son servicios de este tipo. El servidor realiza las llamadas a estos métodos en el cliente sin esperar una respuesta.

Se crean dos interfaces, una correspondiente al cliente (mmsReport) y otra correspondiente al servidor (mms). Estas dos interfaces se han definido en el mismo contexto de nombrado, en el módulo *mmsapp*. El mecanismo para la definición de métodos de estas interfaces cumple:

- Cada método corresponde a un servicio definido en el estándar MMS.

- El parámetro de entrada de un método corresponde a la traducción a IDL del tipo de dato definido en ASN.1 correspondiente a la estructura que transporta la petición de este servicio.

- El resultado de un método correspondiente a un servicio confirmado es la traducción a IDL del tipo de dato definido en ASN.1 correspondiente a la estructura que transporta la respuesta a este servicio.

- En cada método de invocación síncrona se especifica la excepción ServiceErrorType, que da la posibilidad de informar al proceso que ha realizado la llamada de la ocurrencia de una situación excepcional. La estructura ServiceErrorType es la traducción a IDL del tipo de dato especificado en ASN.1 ServiceError.

- Para los métodos correspondientes a los servicios no confirmados el resultado es void.

De esta forma, el método correspondiente al servicio confirmado de petición de estado del VMD es:

Status_ResponseType Status(in Status_RequestType request) raises(ServiceErrorType);

Los demás servicios confirmados son traducidos de una forma similar, teniendo en cuenta el nombre del servicio correspondiente en cada caso.

El método correspondiente al servicio no confirmado de envío de estado no solicitado es:

oneway void UnsolicitedStatus(in UnsolicitedStatus_RequestType request);

Los demás servicios no confirmados son traducidos de una forma similar, teniendo en cuenta el nombre del servicio correspondiente en cada caso.

6.3 Modelo de comunicación

Para la adaptación de MMS a CORBA hay que tener en cuenta el modelo de comunicación implantado en MMS.

El protocolo MMS es orientado a conexión. La conexión en el nivel de aplicación es llamada asociación. Cuando un cliente quiere comunicar con un servidor, se debe

establecer una asociación entre estas dos entidades de aplicación. Debido a que en un sistema CORBA, el ORB oculta el uso de una conexión, esta característica no es necesaria. Los objetos remotos son localizados y accedidos por mecanismos internos del ORB.

Otra característica del protocolo MMS a tener en cuenta es su modelo de interacción asíncrona. Este modelo permite al cliente enviar varias peticiones de forma concurrente, sin tener que esperar la respuesta de una petición para poder enviar la siguiente petición. La petición es identificada por un identificador de petición que es enviado en la petición. En CORBA este identificador no es necesario ya que es el propio ORB el que se encarga de hacer corresponder la petición con la respuesta.

Debido a que una petición de servicio MMS se traduce a IDL como una invocación a un método realizada de forma síncrona, hay que considerar si se puede o no mantener la concurrencia de las peticiones.

En la parte del cliente, CORBA no contempla la ejecución simultánea de varias llamadas a métodos. Por lo tanto, en la parte del cliente es la propia aplicación la que se debe encargar de la concurrencia. Por ejemplo la aplicación podría ejecutar un nuevo hilo o proceso para cada petición que se quiera realizar de forma concurrente.

En la parte del servidor, el ORB tiene mecanismos propios para ejecutar de forma concurrente varias peticiones. Es el Adaptador de Objetos el que se puede encargar de activar un nuevo proceso o hilo por cada petición, esto ocurre en el caso de que el Adaptador de Objetos funcione mediante una política de un servidor por método como se ha explicado en el capítulo correspondiente a CORBA.

6.4 Traducción a IDL

Para la traducción de MMS, que en el estándar está especificado en ASN.1, a IDL, se ha seguido el documento "Inter-domain Management: Specification Translation" [Ope97] generado por el grupo de trabajo JIDM (Joint Inter-Domain Management), compuesto por miembros del Grupo Abierto (The Open Group) y del Foro de Gestión de Red (NMF, The Network Management Forum). Este grupo aborda la necesidad de proporcionar herramientas que permitan la colaboración entre sistemas de gestión basados en diferentes tecnologías, sobre todo para sistemas basados en OSI, SNMP y CORBA.

En el documento de especificación de traducción, se aborda entre otras cosas la Traducción de ASN.1 a OMG IDL. Para este trabajo de tesis se ha tomado esta es-

pecificación de traducción como referencia. En las secciones siguientes se trata en más profundidad esta traducción, tratando en primer lugar la traducción léxica, a continuación la traducción de tipos primitivos y tipos construidos. Para clarificar los métodos de traducción en algunos casos se han incluido ejemplos de la traducción a IDL de tipos MMS especificados en ASN.1.

6.4.1 Traducción Léxica

Cada carácter en el conjunto de caracteres de ASN.1 se traduce a sí mismo en IDL.

En ASN.1, los nombres para las referencias de tipo, los identificadores , las referencias a valores y las referencias a módulos consisten de una secuencia arbitraria de una o más letras, dígitos y guiones. Las letras y dígitos en los nombres ASN.1 son traducidos directamente conservando las mayúsculas y las minúsculas. El guión ("-") se traduce al carácter subrayado ("_"). De esta forma, los nombres son preservados, aunque hay que tener en cuenta que algunos nombres estarán sujetos a la adición de un sufijo.

Los nombres ASN.1 son diferentes si utilizan mayúsculas o minúsculas, cosa que no ocurre en los identificadores IDL. También hay que tener en cuenta que en ASN.1 hay diferentes espacios de nombrado para las referencias a tipos, identificadores y referencias a valores, mientras que en IDL sólo hay un espacio de nombrado. Por ejemplo, los tipos enumerados crean un nuevo espacio de nombrado en ASN.1 pero no en IDL. Esto hay que tenerlo en cuenta en la traducción, para evitar la colisión de nombres IDL.

También hay que tener en cuenta los conflictos que pueden surgir debido a las palabras reservadas de IDL.

Se introducen unas reglas para que no haya ambigüedad en los nombres, pero el objetivo de la traducción es utilizar el mapeado más simple posible, respetando en lo posible los nombres utilizados en la especificación ASN.1. Las reglas son las siguientes:

- Los identificadores que puedan producir un conflicto en la traducción a IDL, el primero no sufre modificación alguna, al segundo se le añade el sufijo "_1", al tercero el sufijo "_2" y así sucesivamente.
- En las referencias de tipo, la primera es traducida con el sufijo "Type", la segunda y siguientes referencias que se encuentren en el mismo ámbito y que sólo difieran en mayúsculas y minúsculas, se le añade el sufijo "Type<n>" donde n es el contador de tales ocurrencias.
- Las referencias a valores serán manejadas como identificadores.

Tipo ASN.1	Tipo IDL
BOOLEAN	typedef boolean ASN1_BOOLEAN
INTEGER	typedef long ASN1_Integer
REAL	typedef double ASN1_Real
NULL	typedef char ASN1_Null; const ASN1_Null ASN1_NullValue= '\x00'
ENUMERATED	enum<enumName> {<elemn1> , ... ,<elemnn> };
BIT STRING	typedef sequence<octet> ASN1_BitString;
OCTET STRING	typedef sequence<octet> ASN1_OctetString;
IA5 STRING	typedef string ASN1_IA5String;
ISO646 STRING	typedef string ASN1_ISO646String;
NUMERIC STRING	typedef string ASN1_NumericString;
PRINTABLE STRING	typedef string ASN1_PrintableString;
TELETEXT STRING	typedef string ASN1_TeletextString;
T16 STRING	typedef string ASN1_T16String;
VIDEO STRING	typedef string ASN1_VideoString;
VISIBLE STRING	typedef string ASN1_VisibleString;
GENERAL STRING	typedef sequence<octet> ASN1_GeneralString;
GRAPHIC STRING	typedef sequence<octet> ASN1_GraphicString;
BMP STRING	typedef sequence<unsigned short> ASN1_BMPString;
UNIVERSAL STRING	typedef sequence<unsigned long> ASN1_UniversalString;
OBJECT IDENTIFIER	typedef string ASN1_ObjectIdentifier;
ANY	typedef any ASN1_Any;
ANY DEFINED BY	typedef any ASN1_DefinedAny;
EXTERNAL	struct ASN1_External { ASN1_ObjectIdentifier syntax; ASN1_DefinedAny data_value; };

Tabla 6.1: Mapeado de tipos ASN.1 a tipos IDL

6.4.2 Traducción de los Tipos Primitivos

La tabla 6.1 resume la traducción de los tipos primitivos ASN.1 a tipos IDL.

6.4.3 Tipos construidos

Los tipos construidos son aquellos que están compuestos de otros tipos (construidos o no). En ASN.1 se soportan los constructores CHOICE, SEQUENCE, SET, SEQUENCE OF y SET OF. En los siguientes apartados se define la traducción de estos tipos construidos a IDL. Para más información acerca de estos constructores se puede

consultar "Apéndice B: Notación de Sintaxis Abstracta Uno (ASN.1)".

Para facilitar la manipulación de algunos tipos, las estructuras de datos complejas serán simplificadas mediante la creación de nuevos tipos IDL intermedios.

Tanto CHOICE, SET como SEQUENCE son el resultado de la agregación de otros tipos. Desde el punto de vista de la gramática, son una lista de elementos. Cada elemento está formado por un tipo nombrado ASN.1 y algunas características adicionales.

Tanto SET OF como SEQUENCE OF son el resultado de la agregación de varias instancias del mismo tipo y serán llamadas items.

6.4.4 Tipos compuestos

Cuando los tipos construidos mencionados anteriormente o bien una enumeración se usan para la definición del tipo de un elemento o item, estos son considerados tipos compuestos. Para evitar las declaraciones de tipos anidadas, se extraen y se define un nuevo tipo IDL, creando de esta forma una definición de tipo nombrado IDL.

El nombre del nuevo tipo IDL se forma concatenando el nombre ASN.1 del contenedor con el nombre del elemento o item con la primera letra en mayúscula y por último se le añade el sufijo "Type". Un ejemplo se proporciona en el apartado correspondiente al mapeado del tipo SEQUENCE.

6.4.5 Mapeado del tipo CHOICE

El tipo CHOICE de ASN.1 se traduce al tipo union de IDL. Debido a que las uniones en IDL son discriminadas, se define un tipo enum para el tipo del discriminador. Para el mapeado de este tipo, previamente se deben transformar los tipos compuestos que se encuentran en la lista de alternativas.

El identificador para el tipo del discriminador se forma añadiendo el sufijo "Choice" al identificador traducido del tipo CHOICE. Para cada alternativa, habrá un identificador en el tipo del distriminador. Este identificador traducido es el identificador de la alternativa seguido por el sufijo "Choice".

El tipo union será entonces construido con un caso para cada alternativa etiquetada de acuerdo al tipo enum y con el tipo e identificador de acuerdo a la alternativa.

Para aclarar la forma de traducción del tipo CHOICE se puede considerar el tipo definido en ASN.1 en el estándar MMS *GetVariableAccessAttributes-Request*. La definición es la siguiente:

```
GetVariableAccessAttributes-Request::= CHOICE
    {
    name        [0] ObjectName,
    address     [1] Address
    }
```

Este tipo es una elección con dos alternativas: *name* de tipo *ObjectName* y *address* de tipo *Address*. Se introduce un tipo enumerado en IDL nombrado *GetVariableAccessAttributes_RequestTypeChoice* con un identificador correspondiente a cada una de las dos alternativas de la elección de la forma siguiente:

```
enum GetVariableAccessAttributes_RequestTypeChoice
      {
      nameChoice,
      addressChoice
      };
```

de manera que el tipo obtenido es la siguiente unión discriminada:

```
union GetVariableAccessAttributes_RequestType switch
          (GetVariableAccessAttributes_RequestTypeChoice)
    {
    case nameChoice:      ObjectNameType name;
    case addressChoice:   AddressType address;
    };
```

Como se ha mencionado anteriormente, el discriminador de esta unión discriminada es del tipo enumerado construido anteriormente y las alternativas de la unión corresponden con las alternativas del tipo CHOICE.

6.4.6 Mapeado del tipo SEQUENCE

El tipo SEQUENCE de ASN.1 se traduce a una structura en IDL. Cada elemento de tipo SEQUENCE es mapeado como un miembro de la estructura IDL en el orden en el que aparece en la lista de miembros.

Sin embargo, en la traducción del tipo SEQUENCE se debe tener en cuenta que puede haber componentes opcionales (OPTIONAL) y que los componentes pueden contar con un valor por defecto.

Si un elemento de un tipo SEQUENCE es declarado opcional (OPTIONAL), quiere decir que el componente puede o no estar presente. Esto se traduce a IDL como una unión discriminada con un discriminador de tipo booleano que sólo tiene un valor en el caso de que el discriminante sea True. Este nuevo tipo se nombra añadiendo al tipo del dato opcional "Opt".

En el ejemplo siguiente, se tiene el tipo definido en ASN.1 StoreDomainContent-Request. Este tipo de dato es una sequencia con tres elementos diferentes: domain-Name, fileName y thirdParty. Este último elemento es opcional. La definición en ASN.1 es la siguiente:

```
StoreDomainContent-Request ::= SEQUENCE
    {
    domainName    [0]   IMPLICIT Identifier,
    fileName      [1]   IMPLICIT FileName,
    thirdParty    [2]   IMPLICIT ApplicationReference OPTIONAL
    }
```

La traducción a IDL supone la inclusión de un tipo adicional ApplicationReference-TypeOpt que como se ha comentado anteriormente corresponde a una unión discriminada con el discriminador de tipo booleano y que contiene un valor únicamente si el discriminador es verdad. Este tipo IDL es el siguiente:

```
union ApplicationReferenceTypeOpt switch (boolean) {
   case TRUE: ApplicationReferenceType value;
};
```

y el tipo estructura entonces quedaría de la forma siguiente:

```
struct StoreDomainContent_RequestType
    {
    IdentifierType domainName;
    FileNameType fileName;
    ApplicationReferenceTypeOpt thirdParty;
    };
```

donde cada uno de los elementos del tipo SEQUENCE de ASN.1 se ha traducido a un elemento de la estructura.

La declaración de un valor por defecto en ASN.1 mediante la etiqueta DEFAULT no tiene traducción directa a IDL. Lo que se hace en este caso es simplemente proporcionar este valor como una constante, que será usada por el programador en el dominio CORBA de forma explícita. El tipo de la constante se declara como el tipo mapeado a IDL del componente que tiene un valor por defecto y el nombre de la constante será el nombre de este componente al que se le añade el sufijo "Default".

En el caso del tipo DefineEventCondition-Request se tienen valores por defecto para el elemento priority y severity. La definición en ASN.1 es mostrada a continuación:

```
DefineEventCondition-Request ::= SEQUENCE
    {
    eventConditionName      [0] ObjectName,
    class                   [1] IMPLICIT EC-Class,
    priority                [2] IMPLICIT Priority DEFAULT 64,
    severity                [3] IMPLICIT Unsigned8 DEFAULT 64,
    alarmSummaryReports     [4] IMPLICIT BOOLEAN OPTIONAL,
    monitoredVariable       [6] VariableSpecification OPTIONAL,
    evaluationInterval      [7] IMPLICIT Unsigned32 OPTIONAL
    }
```

La traducción para los valores por defecto para priority y severity es la siguiente:

```
const PriorityType priorityDefault = 64;

const Unsigned8Type severityDefault = 64;
```

Se han introducido dos constantes que reflejan los valores por defecto para estos elementos. La traducción de este tipo entonces queda de esta manera:

```
struct DefineEventCondition_RequestType
    {
    ObjectNameType eventConditionName;
    EC_ClassType class;
    PriorityType priority;
    Unsigned8Type severity;
```

```
    ASN1_BooleanOpt alarmSummaryReports;
    VariableSpecificationTypeOpt monitoredVariable;
    Unsigned32TypeOpt evaluationInterval;
    };
```

A continuación se expone el último ejemplo del tipo SEQUENCE para aclarar lo que se ha explicado en el apartado 6.4.4 referente a los tipos compuestos. En el tipo ASN.1 GetEventConditionAttributes-Response existe un elemento cuyo tipo es un tipo CHOICE por lo que se considera un tipo compuesto:

```
GetEventConditionAttributes-Response ::= SEQUENCE
    {
    mmsDeletable            [0] IMPLICIT BOOLEAN DEFAULT FALSE,
    class                   [1] IMPLICIT EC-Class,
    priority                [2] IMPLICIT Priority DEFAULT 64,
    severity                [3] IMPLICIT Unsigned8 DEFAULT 64,
    alarmSummaryReports     [4] IMPLICIT BOOLEAN DEFAULT FALSE,
    monitoredVariable       [6] CHOICE
        {
        variableReference   [0] VariableSpecification,
        undefined           [1] IMPLICIT NULL
        } OPTIONAL,
    evaluationInterval      [7] IMPLICIT Unsigned32 OPTIONAL
    }
```

La traducción a IDL es la siguiente:

```
enum
  GetEventConditionAttributes_ResponseMonitoredVariableTypeChoice
      {
      variableReferenceChoice,
      undefinedChoice
      };

union GetEventConditionAttributes_ResponseMonitoredVariableType
switch
(GetEventConditionAttributes_ResponseMonitoredVariableTypeChoice)
    {
    case variableReferenceChoice:
         VariableSpecificationType variableReference;
```

```
    case undefinedChoice:
         ASN1_Null undefined;
    };

union GetEventConditionAttributes_ResponseMonitoredVariableTypeOpt
switch (boolean)
      {
      case TRUE:
      GetEventConditionAttributes_ResponseMonitoredVariableType value;
      };

const ASN1_Boolean mmsDeletableDefault = FALSE;

const ASN1_Boolean alarmSummaryReportsDefault = FALSE;

struct GetEventConditionAttributes_ResponseType
    {
    ASN1_Boolean mmsDeletable;
    EC_ClassType class;
    PriorityType prio_rity;
    Unsigned8Type severity;
    ASN1_Boolean alarmSummaryReports;
    GetEventConditionAttributes_ResponseMonitoredVariableTypeOpt
                                                  monitoredVariable;
    Unsigned32TypeOpt evaluationInterval;
    };
```

6.4.7 Mapeado del tipo SEQUENCE OF

Este tipo indica una lista ordenada de elementos del mismo tipo. Se mapea al tipo IDL sequence. Primero se define un nuevo tipo para los elementos de la secuencia. El nuevo tipo es nombrado con el nombre del tipo ASN.1 seguido por el sufijo Item y el sufijo Type. A continuación se define el tipo como una secuencia de elementos del tipo anteriormente mencionado.

Para el tipo GetNameList-Response definido en ASN.1 como sigue:

```
GetNameList-Response ::= SEQUENCE
    {
    listOfIdentifier    [0] IMPLICIT SEQUENCE OF Identifier,
    moreFollows         [1] IMPLICIT BOOLEAN DEFAULT TRUE
```

```
}
```

la traducción es la siguiente:

```
const ASN1_Boolean moreFollowsDefault = TRUE;

typedef IdentifierType
GetNameList_ResponselistOfIdentifierItemType;

struct GetNameList_ResponseType {
 sequence<GetNameList_ResponselistOfIdentifierItemType>
 listOfIdentifier;
 ASN1_Boolean moreFollows;
};
```

6.4.8 Mapeado del tipo SET

El mapeado del tipo SET es idéntico al mapeado del tipo SEQUENCE, ya que este tipo corresponde a un registro de elementos no ordenados, que no tiene correspondencia con ningún tipo IDL.

6.4.9 Mapeado del tipo SET OF

Igual que en el caso anterior, el mapeado del tipo SET OF es idéntico al mapeado del tipo SEQUENCE OF.

6.4.10 Mapeado del Tipo Restringido

Cuando se usa una restricción de tamaño, bien para el tipo SET OF o SEQUENCE OF, se mapea al tipo IDL sequence, en la que se indica una limitación para el tamaño, el límite superior de la secuencia IDL es determinada por el límite superior de los valores enteros permitidos que se especifican en la restricción.

6.4.11 Mapeado de Tipos Recursivos

ASN.1 permite la definición de tipos recursivos, sin embargo en IDL no están soportados, excepto con las secuencias. De todas formas sólo se puede utilizar la recursión directa. La recursión indirecta es explícitamente no soportada.

Las definiciones de tipos recursivos se mapean convirtiendo la referencia del tipo a un secuencia IDL limitada de tamaño 1. Si una referencia recursiva se produce directamente en SET OF o SEQUENCE OF, entonces se omite el límite. Las definiciones de tipos recursivos ASN.1 son expandidas en el mismo tipo IDL resultado del mapeo.

Un ejemplo de tipo ASN.1 recursivo extraído de MMS es el siguiente:

```
Data ::= CHOICE
   {
   array                [1]  IMPLICIT SEQUENCE OF Data,
   structure            [2]  IMPLICIT SEQUENCE OF Data,
   boolean              [3]  IMPLICIT BOOLEAN,
   bit-string           [4]  IMPLICIT BIT STRING,
   integer              [5]  IMPLICIT INTEGER,
   unsigned             [6]  IMPLICIT INTEGER,
   floating-point       [7]  IMPLICIT FloatingPoint,
   real                 [8]  IMPLICIT REAL,
   octet-string         [9]  IMPLICIT OCTET STRING,
   visible-string       [10] IMPLICIT VisibleString,
   binary-time          [12] IMPLICIT TimeOfDay,
   bcd                  [13] IMPLICIT INTEGER,
   booleanArray         [14] IMPLICIT BIT STRING
   }
```

La traducción a IDL teniendo en cuenta que es un tipo recursivo es la siguiente:

```
enum DataChoice
      {
      arrayChoice_1,
      structureChoice_1,
      booleanChoice_1,
      bit_stringChoice_1,
      integerChoice_1,
      unsignedChoice_1,
```

```
      floating_pointChoice_1,
      realChoice_1,
      octet_stringChoice_1,
      visible_stringChoice_1,
      binary_timeChoice_1,
      bcdChoice_1,
      booleanArrayChoice
      };

union DataType switch(DataChoice)
      {
      case arrayChoice_1: sequence<DataType> array;
      case structureChoice_1: sequence<DataType> structure;
      case booleanChoice_1: ASN1_Boolean boolean_1;
      case bit_stringChoice_1: ASN1_BitString bit_string;
      case integerChoice_1: ASN1_Integer integer;
      case unsignedChoice_1: ASN1_Integer unsigned_1;
      case floating_pointChoice_1: FloatingPointType floating_point;
      case realChoice_1: ASN1_Real real;
      case octet_stringChoice_1: ASN1_OctetString octet_string;
      case visible_stringChoice_1: ASN1_VisibleString visible_string;
      case binary_timeChoice_1: TimeOfDayType binary_time;
      case bcdChoice_1: ASN1_Integer bcd;
      case booleanArrayChoice: ASN1_BitString booleanArray;
      };
```

Los tipos recursivos indirectos en cada caso se han simplificado en lo posible eliminando la recursión. Por ejemplo el siguiente tipo especificado en el estándar MMS en ASN.1 como sigue:

```
VariableSpecification ::= CHOICE
    {
    name                          [0] ObjectName,
    address                       [1] Address,
    variableDescription           [2] IMPLICIT SEQUENCE
        {
        address          Address,
        typeSpecification   TypeSpecification
        },
    scatteredAccessDescription  [3] IMPLICIT ScatteredAccessDescription,
    invalidated                 [4] IMPLICIT NULL
    }
```

```
ScatteredAccessDescription ::= SEQUENCE OF SEQUENCE
    {
    componentName           [0] IMPLICIT Identifier OPTIONAL,
    variableSpecification   [1] VariableSpecification,
    alternateAccess         [2] IMPLICIT AlternateAccess OPTIONAL
    }
```

En IDL se ha eliminado la cuarta posibilidad del tipo VariableSpecification. De esta forma se elimina la recursión mutua. Si es necesario utilizar la descripción de un acceso extendido se puede definir un objeto del tipo Acceso Extendido y luego incluir en la especificación de la variable el nombre de este objeto. El resultado en IDL es el siguiente:

```
enum VariableSpecificationTypeChoice
      {
      nameChoice,
      addressChoice,
      variableDescriptionChoice,
      invalidatedChoice
      };

struct VariableSpecificationVariableDescriptionType
      {
      AddressType  address;
      TypeSpecificationType typeSpecification;
      };

union VariableSpecificationType switch
     (VariableSpecificationTypeChoice)
      {
      case nameChoice: ObjectNameType name;
      case addressChoice: AddressType address;
      case variableDescriptionChoice:
           VariableSpecificationVariableDescriptionType
           variableDescription;
      case invalidatedChoice: ASN1_Null invalidated;
      };

struct ScatteredAccessDescriptionItemType
      {
      IdentifierTypeOpt componentName;
```

```
      VariableSpecificationType variableSpecification;
      AlternateAccessTypeOpt alternateAccess;
      };

typedef sequence<ScatteredAccessDescriptionItemType>
        ScatteredAccessDescriptionType;
```

6.5 Referencia a la Aplicación

Aunque la gestión de la asociación no es implementada debido al hecho de que son gestionadas por el ORB, se deben hacer algunas consideraciones.

En algunos casos, en los objetos y servicios del modelo de eventos, es importante poder indicar una asociación específica. Por ejemplo, el objeto Registro de Evento tiene un atributo que es una referencia a la aplicación que debe ser advertida mediante un servicio Event Notification cuando el estado de una variable cambia. De esta forma, el servidor MMS debe ser capaz de indicar el evento a un determinado cliente. En el estándar MMS la referencia a la aplicación es definida como se especifica a continuación:

```
ApplicationReference ::= SEQUENCE {
  ap-title
      [0] AP-title                 OPTIONAL,
  ap-invocation-id
      [1] AP-invocation-identifier OPTIONAL,
  ae-qualifier
      [2] AE-qualifier             OPTIONAL,
  ae-invocation-id
      [3] AE-invocation-identifier OPTIONAL
}
```

Pero esta información es sólo significativa en un entorno OSI. En CORBA, la siguiente referencia a la aplicación se ha utilizado:

```
enum ApplicationReferenceTypeChoice {
   name,
   reference
};
```

```
union ApplicationReferenceType
      switch (ApplicationReferenceTypeChoice)
{
  case name:
      IdentifierType ap_name;
  case reference:
      mmsReport ap_reference;
};
```

La referencia a la aplicación se ha traducido a una unión discriminada en IDL. El discriminador es un enumerado con dos identificadores *name* y *reference*. Dependiendo del valor del discriminador, la referencia a la aplicación será un identificador que corresponde con el nombre de la aplicación o una referencia al objeto mmsReport. Esta referencia es por lo tanto el nombre de la aplicación o bien la referencia a un objeto CORBA. Teniendo esta referencia, el servidor MMS puede comunicar con el cliente directamente por medio del ORB. En otro caso, teniendo el nombre, puede obtener esta referencia preguntando al Servicio de Nombrado.

Capítulo 7

Modelos para la Comunicación de Eventos

Como ya se ha comentado en capítulos anteriores, el objeto gestionado debe tener la capacidad de generar notificaciones de ocurrencias excepcionales en el dispositivo, para evitar de esta forma el sondeo constante por parte del objeto gestor.

El protocolo MMS contempla la posibilidad de la comunicación de notificaciones de eventos en su modelo de eventos ya introducido en el capítulo correspondiente a este protocolo.

En este capítulo se presentan dos modelos diferentes para permitir la comunicación de eventos producidos en el Dispositivo de Fabricación Virtual a objetos CORBA. En primer lugar se detalla el modelo de eventos de MMS así como la traducción de las primitivas de este modelo a IDL y a continuación se comentan los dos modelos propuestos, el primero se ajusta al modelo de comunicación de eventos de MMS y el segundo introduce la utilización del canal de eventos definido en el servicio de eventos CORBA.

7.1 Modelo de Eventos de MMS

Como ya se comentó en el capítulo correspondiente a MMS, el modelo de eventos introduce tres clases de objetos diferentes: la clase Event Condition, la clase Event Enrollment y la clase Event Action.

A continuación se van a detallar estos objetos incluyendo la plantilla para la definición de clases de objetos utilizada en MMS. También se van a explicar los distintos servicios incluidos en este modelo.

Condición de Evento

Un objeto *Condición de Evento* (Event Condition) representa el estado actual de una condición en el VMD que puede producir una notificación de evento. No define la acción a realizar. En MMS no se define el mapeado del valor de la variable y el estado de la variable condición. Existen dos clases de objetos condición de evento:

- Network Triggered: El cliente MMS dispara el evento mediante el servicio Trigger-Event.
- Monitored: El objeto Event Condition tiene una variable booleana asociada. El cambio de estado de esta variable booleana produce la notificación de evento.

La plantilla en MMS correspondiente al objeto Condición de Evento que define el conjunto de atributos de los objetos de esta clase es la siguiente:

```
Object: Event Condition
   Key Attribute:    Event Condition Name
   Attribute:        MMS Deletable (TRUE, FALSE)
   Attribute:        Event Condition Class ( NETWORK-TRIGGERED,
                                             MONITORED)
   Attribute:        State (DISABLED, IDLE, ACTIVE)
   Attribute:        Priority
   Attribute:        Severity
   Attribute:        Additional Detail
   Attribute:        List Of Event Enrollment Reference
   Constraint        Event Condition Class = MONITORED
                     Attribute:          Enabled (TRUE, FALSE)
                     Attribute:          Alarm Summary Reports
                                         (TRUE, FALSE)
                     Attribute:          Monitored Variable Reference
                     Attribute:          Evaluation Interval
                     Attribute:          Time Of Last Transition
                                         To Active
                     Attribute:          Time Of Last Transition
                                         To Idle
```

Los atributos son los siguientes:

- Nombre de la Condición de Evento (Event Condition Name): Este atributo identifica de forma única al objeto.
- Borrable mediante MMS (MMS Deletable): Indica si el objeto puede ser borrado mediante el servicio MMS de borrado de Condición de Evento.
- Clase de la Condición de Evento (Event Condition Class): Indica la clase del objeto Condición de Evento. La clase es NETWORK-TRIGGERED o bien MONITORED correspondiendo con las dos clases mencionadas anteriormente.
- Estado (State): Estado actual del objeto. El estado puede ser deshabilitado, ocioso o activo (DISABLED, IDLE, ACTIVE).
- Prioridad (Priority): Indica la importancia del objeto Condición de Evento relativo a otros objetos de este tipo. El valor de la prioridad determinará el orden en el que se procesen las transiciones de las condiciones de eventos. Tiene un rango de 0 a 127, siendo 0 la prioridad más alta y 64 el valor para la prioridad "normal".
- Severidad (Severity): Representa el efecto del evento en el proceso que está siendo controlado. El valor de la severidad tiene un rango de 0 a 127, siendo 0 la severidad más alta y 64 el valor para la severidad "normal". El comportamiento del Dispositivo de Fabricación Virtual con respecto a la influencia del valor de este atributo es una cuestión local.
- Detalle Adicional (Additional Detail): Información de estado específica del dispositivo.
- Lista de Referencias a Registros de Eventos (List Of Event Enrollment Reference): Este atributo contiene una lista de referencias a cero o más objetos Registro de Evento.
- Si el objeto Condición de Evento es monitorizado (MONITORED), entonces se tienen también los siguientes atributos:
  - Habilitado (Enabled): Especifica si (TRUE) o no (FALSE) los cambios en los valores de la variable referenciada por este objeto causará la invocación del procedimiento de procesamiento de la transición de evento en los objetos Registro de Evento referenciado en el atributo Lista de Referencias a Registros de Eventos.
  - Informes de Resumen de Alarmas (Alarm Summary Reports): Este atributo indica si (TRUE) o no (FALSE) se considera este objeto para el tratamiento del servicio de obtención de resúmenes de alarmas.

- Referencia a la Variable Monitorizada (Monitored Variable Reference): Es una referencia a una variable de tipo booleano.
- Intervalo de Evaluación (Evaluation Interval): Especifica el tiempo máximo aceptable para la determinación del valor del atributo de estado.
- Tiempo de Última Transición a Activo (Time Of Last Transition To Active): Registra el tiempo de la última transición detectada del estado del objeto Condición de Evento a activo.
- Tiempo de Última Transición a Ocioso (Time Of Last Transition To Idle): Registra el tiempo de la última transición detectada del estado del objeto Condición de Evento a ocioso.

Registro de Evento

El objeto *Registro de Evento* (Event Enrollment) identifica a quién notificar acerca de la ocurrencia de un evento. Este objeto como ya se comentó en el capítulo correspondiente a MMS enlaza todos los elementos del modelo de gestión de eventos MMS. Representa una petición de un cliente MMS de ser notificado acerca de los cambios en el estado de una condición de evento. Se hace referencia a la condición de evento, a la acción de evento (opcional) y al cliente al que hay que enviar la notificación.

Existen dos tipos de objetos Registro de Evento:

- Modificador (MODIFIER): Representa un Registro de Evento temporal, creado como el resultado de la recepción de una petición de servicio confirmada modificada mediante el modificador de conexión a Condición de Evento. Esto permite al cliente retrasar la ejecución de un servicio hasta que se produzca una transición en el estado de un objeto Condición de Evento.
- Notificación (NOTIFICATION): Representa un objeto definido o predefinido explícitamente. Requiere que se realice el procedimiento de procesamiento de la transición de evento cuando se detecte las transiciones especificadas.

La plantilla en MMS correspondiente al objeto Registro de Evento es la siguiente:

```
Object: Event Enrollment
        Key Attribute:    Event Enrollment Name
        Attribute:        MMS Deletable (TRUE, FALSE)
        Attribute:        Enrollment Class ( MODIFIER, NOTIFICATION)
```

```
Attribute:          Event Condition Reference
Attribute:          Event Condition Transitions
Attribute:          Application Association Local Tag
Constraint          Enrollment Class = MODIFIER
                    Attribute:          Invoke ID
                    Attribute:          Remaining Acceptable
                                        Delay
Constraint          Enrollment Class = NOTIFICATION
                    Attribute:          Notification Lost
                                        (TRUE, FALSE)
                    Attribute:          Event Action Reference
                    Attribute:          Acknowledgement
                                        Event Condition Reference
                    Attribute:          Duration
                    Attribute:          Client Application
                    Attribute:          Additional Detail
                    Attribute:          Alarm Acknowledgement
                                        Rule
                    Attribute:          Time Active Acknowledged
                    Attribute:          Time Idle Acknowledged
                    Attribute:          State
```

En Esta plantilla se definen los atributos de este objeto que son los siguientes:

- Nombre del Registro de Evento (Event Enrollment Name): Se utiliza para identificar el objeto.
- Borrable mediante MMS (MMS Deletable): Indica si (TRUE) o no (FALSE) puede ser borrado el objeto mediante el servicio de borrado de Registro de Evento.
- Clase de Registro (Enrollment Class): Indica si el tipo de Registro es MODIFIER o NOTIFICATION.
- Referencia a Condición de Evento (Event Condition Reference): Referencia al objeto Condición de Evento cuya transición del atributo Estado provoca la invocación del procedimiento para el procesamiento de la transición de evento para este objeto Registro de Evento.
- Transiciones de la Condición de Evento (Event Condition Transitions): Es una lista de todas las transiciones del objeto Condición de Evento que hacen que se invoque el procedimiento de procesamiento de la transición de evento.
- Etiqueta Local de la Asociación de la Aplicación (Application Association Local Tag): Identifica localmente a la aplicación a la cual hay que notificar el evento.

- Si la clase de registro es MODIFIER entonces se cuenta con los siguientes atributos:
  - ID de la invocación (Invoke ID): Contiene la identificación de la invocación del servicio modificado.
  - Retraso Aceptable (Remaining Acceptable Delay): Especifica el tiempo en segundos que el cliente que pidió el servicio modificado está dispuesto a esperar a que ocurra la transición de la condición de evento especificada.
- Si la clase de registro es NOTIFICATION entonces se cuenta con estos otros atributos:
  - Pérdida de Notificación (Notification Lost): Este atributo tendrá el valor verdadero en el caso en que limitaciones de recursos en el VMD no permita la ejecución normal del procedimiento de procesamiento de transición de evento o bien no se pueda establecer comunicación con la aplicación que tiene que ser notificada.
  - Referencia a Acción de Evento (Event Action Reference): Este atributo puede contener una referencia a un objeto Acción de Evento correspondiente al servicio que debe ser ejecutado cuando se produzca el evento y cuyo resultado debe ser incluido en la notificación de evento.
  - Referencia a la Condición de Evento de Reconocimiento (Acknowledgement Event Condition Reference): En este atributo se puede indicar la referencia a un objeto Condición de Evento del tipo network-triggered que servirá como reconocimiento de la Condición de Evento.
  - Duración (Duration): Indica la duración del objeto Registro de Evento y puede tener el valor CURRENT indicando que el objeto es definido para la asociación de la aplicación en la que el objeto fue definido o el valor PERMANENT indicando que el objeto permanecerá durante toda la vida del Dispositivo de Fabricación Virtual o hasta que sea expresamente borrado.
  - Aplicación Cliente (Client Application): Contiene la identificación de la aplicación cliente registrada.
  - Detalle Adicional (Additional Detail): Contiene información de estado adicional específica del dispositivo y la aplicación.
  - Regla de Reconocimiento de Alarma (Alarm Acknowledgement Rule): Este atributo indica el nivel de reconocimiento requerido para el servicio de notificación de evento y se determina mediante este atributo si el objeto Registro de Evento será o no incluido en los sumarios de registros de alarmas.
  - Tiempo de Activo Reconocido (Time Active Acknowledged): Registra el tiempo en el que se ha recibido un reconocimiento para la transición detectada más recientemente del objeto Condición de Evento al estado activo.

- Tiempo de Ocioso Reconocido (Time Idle Acknowledged): Registra el tiempo en el que se ha recibido un reconocimiento para la transición detectada más recientemente del objeto Condición de Evento al estado ocioso.
- Estado (State): Mantiene el estado actual del objeto Registro de Evento.

Acción de Evento

El objeto *Acción de Evento* (Event Action) permite especificar una acción que debe ser realizada cuando se produce el evento. Representa la acción que el VMD realizará cuando el estado de la condición de evento varíe. Es opcional. Es definida como una petición de servicio confirmado (Start, Stop, Read, etc). La respuesta es incluida en la petición de servicio EventNotification que es enviado al cliente. No pueden ser utilizados los servicios no confirmados ni los servicios que deben ser utilizados en conjunción con otros servicios.

La plantilla que define en MMS los atributos correspondientes a este objeto es la siguiente:

```
Object: Event Action
        Key Attribute:     Event Action Name
        Attribute:         MMS Deletable (TRUE, FALSE)
        Attribute:         Confirmed Service Request
        Attribute:         List Of Modifier
        Attribute:         List Of Event Enrollment Reference
        Attribute:         Additional Detail
```

Los atributos especificados son:

- Nombre de Acción de Evento (Event Action Name): Identifica de forma única al objeto.
- Borrable mediante MMS (MMS Deletable): Indica si el objeto puede ser (TRUE) o no (FALSE) borrado mediante el servicio correspondiente de borrado de Acción de Evento.
- Petición de Servicio Confirmado (Confirmed Service Request): Identifica el servicio que se ejecutará durante el procesamiento de la transición de evento. Los parámetros están incluidos en este atributo y el resultado se enviará al cliente cuando se realice una notificación de evento.

- Lista de Modificadores (List Of Modifier): Incluye la lista de modificadores que será aplicada a cada ejecución de la acción de evento.
- Lista de Referencias a Registros de Eventos (List Of Event Enrollment Reference): Mantiene una lista de referencias a objetos del tipo Registro de Evento que actualmente están referenciando a este objeto.
- Detalle Adicional (Additional Detail): Contiene información de estado específica del dispositivo o de la aplicación.

Servicios

A continuación se comentan brevemente los distintos servicios definidos en el modelo de eventos de MMS:

- DefineEventCondition: Proporciona al cliente un mecanismo para requerir la creación de un objeto Condición de Evento en el Dispositivo de Fabricación Virtual.
- DeleteEventCondition: El propósito de este servicio es permitir al usuario borrar uno o más objetos Condición de Evento.
- GetEventConditionAttributes: Permite obtener los atributos del objeto Condición de Evento.
- ReportEventConditionStatus: Permite obtener el estado del objeto Condición de Evento.
- AlterEventConditionMonitoring: Sirve para alterar los atributos Habilitado, Prioridad, Informe Resumen de Alarma e Intervalo de Evaluación del objeto Condición de Evento.
- TriggerEvent: Este servicio se utiliza para disparar el evento asociado a un objeto Condición de Evento del tipo Network-triggered.
- DefineEventAction: Proporciona al cliente un mecanismo para requerir la creación de un objeto Acción de Evento en el Dispositivo de Fabricación Virtual.
- DeleteEventAction: El propósito de este servicio es permitir al usuario borrar uno o más objetos Acción de Evento.
- GetEventActionAttributes: Permite obtener los atributos del objeto Acción de Evento.

- ReportEventActionStatus: Se utiliza para obtener el número de objetos Registro de Evento que especifican actualmente un determinado objeto Acción de Evento.

- DefineEventEnrollment: Proporciona al cliente un mecanismo para requerir que el Dispositivo de Fabricación Virtual añada a la aplicación que realiza la petición de servicio o alguna otra aplicación a la lista de usuarios para los cuales se ejecutará el procedimiento de procesamiento de transición de eventos como un resultado de una transición especificada de un objeto Condición de Evento.

- DeleteEventEnrollment: El propósito de este servicio es permitir al usuario borrar uno o más objetos Registro de Evento.

- GetEventEnrollmentAttributes: Permite obtener los atributos del objeto Registro de Evento.

- ReportEventEnrollmentStatus: Permite obtener el estado de un objeto Registro de Evento.

- AlterEventEnrollment: Proporciona un medio que permite al cliente pedir al Dispositivo de Fabricación Virtual reemplazar los valores de los atributos Event Condition Transitions, Alarm Acknowledgement Rule o ambos de un objeto Registro de Evento existente.

- EventNotification: Este servicio permite al Dispositivo de Fabricación Virtual notificar a un cliente registrado acerca de la ocurrencia de una transición de estado asociado con un objeto Condición de Evento.

- AcknowledgeEventNotification: Este servicio permite al cliente MMS informar al Dispositivo de Fabricación Virtual de que ha recibido la notificación de evento.

- GetAlarmSummary: Mediante este servicio el cliente MMS puede pedir información referente al estado de objetos Condición de Evento así como de los objetos Registro de Evento asociados a éstos.

- GetAlarmEnrollmentSummary: Mediante este servicio el cliente MMS puede pedir información acerca del estado de las alarmas de los objetos Registro de Evento así como de los objetos Condición de Evento asociados a éstos.

7.2 Traducción de Primitivas de MMS

Al objeto computacional VMD se le añaden los servicios para el tratamiento de eventos [Ari98a]. La forma de adaptar estos servicios especificados en ASN.1 a IDL es la especificada en el capítulo anterior referente a la adaptación de MMS a CORBA.

En este trabajo de tesis no se han tenido en cuenta los modificadores, por simplicidad y debido a que no se ajustan al carácter síncrono de la peticiones CORBA. Sin embargo no se descarta para trabajos futuros, la adaptación de esta propuesta para soportar modificadores.

A continuación se muestra el ejemplo del servicio TriggerEvent. Se trata la forma de traducir las estructuras de datos necesarias desde ASN.1 a IDL.

Los datos especificados en el protocolo MMS para llevar a cabo una petición y devolver la respuesta para el servicio TriggerEvent son los siguientes:

```
Identifier ::= VisibleString

ObjectName ::= CHOICE
   {
   vmd-specific
         [0] IMPLICIT Identifier,
   domain-specific
         [1] IMPLICIT SEQUENCE
      {
      domainId        Identifier,
      itemId          Identifier
      },
   aa-specific
         [2] IMPLICIT Identifier
   }

Unsigned8 ::= INTEGER
Priority ::= Unsigned8

TriggerEvent-Request ::= SEQUENCE
   {
   eventConditionName
          [0] ObjectName,
   priority
          [1] IMPLICIT Priority OPTIONAL
   }

TriggerEvent-Response ::= NULL
```

Las estructuras correspondientes en IDL obtenidas por medio de la traducción siguiendo la especificación del Grupo Abierto son las siguientes:

```
typedef string ASN1_VisibleString;
```

```
typedef ASN1_VisibleString IdentifierType;
typedef long ASN1_Integer;
typedef ASN1_Integer Unsigned8Type;
typedef Unsigned8Type PriorityType;

union PriorityTypeOpt switch (boolean) {
   case TRUE: PriorityType value;
};

enum ObjectNameTypeChoice {
   vmd_specificChoice,
   domain_specificChoice,
   aa_specificChoice };

struct domain_specificItemType{
   IdentifierType domainId;
   IdentifierType itemId;
};

struct aa_specificItemType{
   IdentifierType aaId;
   IdentifierType itemId;
};

union ObjectNameType switch (ObjectNameTypeChoice) {
   case vmd_specificChoice:
      IdentifierType vmd_specific;
   case domain_specificChoice:
      domain_specificItemType domain_specific;
   case aa_specificChoice:
      aa_specificItemType aa_specific;
};

struct TriggerEvent_RequestType {
  ObjectNameType eventConditionName;
  PriorityTypeOpt priority;
};
```

El método en el interfaz especificado en IDL es:

```
void TriggerEvent(TriggerEvent_RequestType)
                  raises(ServiceErrorType):
```

Una vez la interfaz del VMD para gestionar eventos MMS es definida en IDL, los

servicios del objeto CORBA mms para la gestión de eventos pueden ser accedidos a través del ORB desde cualquier parte del sistema distribuido de fabricación.

Para que el cliente pueda recibir notificaciones de evento, debe contar con un objeto computacional que incluya este servicio.

7.3 Comunicación de Eventos

El modelo para la comunicación de eventos se basa en seguir el mecanismo MMS para la notificación de eventos, usando los servicios del objeto CORBA mms.

Un cliente que quiera escuchar notificaciones de eventos tendrá en primer lugar que crear los registros de evento en el VMD necesarios para ser notificado de estas ocurrencias.

La creación de estos registros se lleva a cabo mediante el servicio CreateEventEnrolment. En la creación se indica el evento del que se quiere ser informado y la aplicación cliente a la que hay que comunicar el evento.

Cuando se produzca el evento, la aplicación cliente será notificada de la ocurrencia de este evento. Para ello, el cliente debe ser un objeto CORBA que incluya el servicio para la notificación de eventos obtenido de la traducción del servicio correspondiente en MMS.

Cada cliente debe crear los registros de eventos que crea conveniente para ser informado de las ocurrencias de eventos en los que está interesado.

Cada vez que se produce un evento, el VMD debe informar a todos los clientes que hayan registrado su interés.

7.4 Uso del Canal de Eventos de CORBA

Otra forma propuesta [Ari99b] para la notificación de eventos al cliente es mediante el uso del canal de eventos de CORBA. El canal es el encargado de difundir estas notificaciones. En este caso, el objeto VMD no tiene que conocer todas las aplicaciones que están interesadas en una determinada ocurrencia de evento, sino que es el canal de eventos de CORBA el que se encarga de mantener el enlace con todas las aplicaciones

consumidoras de estas notificaciones de eventos.

La decisión de usar o no el canal de eventos se hará en base a las características propias de la aplicación distribuida. Será aconsejable sobre todo en aquellas aplicaciones donde haya muchos clientes interesados en los mismos eventos. Con la utilización del canal se simplifica el proceso de los registros de los clientes, ya que se evita el que cada cliente tenga que crear un registro por evento. El cliente se limita a darse de alta en el canal de eventos como consumidor. Se simplifica también la información almacenada en el VMD referente a estos registros.

A continuación se introduce el servicio de eventos de CORBA, para comprender el modelo de comunicación de eventos de este servicio, los distintos objetos sobre los que se basa el modelo y la forma de funcionamiento para la notificación de eventos. También se comenta el elemento proxy introducido para la notificación de eventos MMS a objetos CORBA.

7.4.1 Servicio de Eventos de CORBA

Los requerimientos CORBA son generalmente dirigidos a un objeto específico mediante métodos de invocación síncrona, donde la petición es dirigida a un objeto particular. Es este modelo tanto el cliente como el servidor deben estar disponibles. El esquema de funcionamiento se muestra en la figura 7.1.

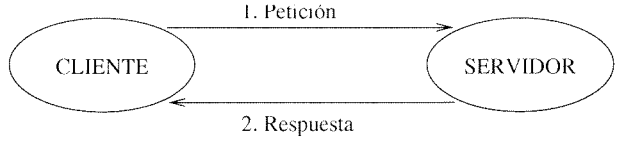


Figura 7.1: Método de Invocación Síncrona

En algunos casos, se necesita un modelo de comunicación entre objetos que permita enviar información de una forma asíncrona. Por ejemplo si una herramienta de administración de sistemas quiere conocer cuando se llena el disco o si un objeto lista de propiedades está asociado a un objeto aplicación y el objeto aplicación quiere ser informado de los cambios, etc. Por este motivo se introduce el servicio de comunicación de eventos en CORBA. Mediante este servicio un objeto puede ser informado de una forma asíncrona de ocurrencias de eventos en los que está interesado.

En el servicio de eventos se definen dos roles bien diferenciados:

- Proveedores: Que producen eventos y los envían a los consumidores interesados en ellos.

- Consumidores: Que registran su interés en los eventos y se encargan de escuchar estos eventos.

Los eventos son comunicados entre los proveedores y los consumidores usando requerimientos estándar CORBA. Para evitar la relación directa de un objeto con el rol de consumidor con un objeto con el rol de proveedor y poder de esta forma introducir la comunicación con un grupo de consumidores y proveedores, se incluye en el modelo un objeto CORBA que se encarga de desacoplar la comunicación de los objetos con estos dos roles.

Este objeto es el Canal de Eventos que proporciona una comunicación muchos a muchos. Es decir, Puede recibir datos de eventos de varios proveedores y puede enviar datos de eventos a varios consumidores, permitiendo de esta forma una comunicación más flexible.

Se pueden utilizar dos tipos diferentes de eventos:

- Eventos tipados: El proveedor y el consumidor comparten conocimiento de las interfaces dependientes de la aplicación.
- Eventos no tipados (genéricos): El evento es del tipo Any definido en CORBA. El tipo Any puede mantener un tipo IDL o un tipo definido por el usuario. Los proveedores y consumidores deben implícitamente conocer el tipo real de evento que debería ser recibido.

Modelos de Distribución

La distribución de eventos se puede realizar siguiendo dos modelos:

- Modelo pull: En este modelo los proveedores esperan pasivamente peticiones de eventos del canal mientras que los consumidores piden activamente eventos del canal.
- Modelo push: En este caso, los proveedores envían activamente los eventos al canal mientras que los consumidores reciben pasivamente eventos del canal.

En las figuras 7.2 y 7.3 se muestran respectivamente el modelo pull y el modelo push. El sentido de las flechas no indica el sentido del flujo de información del evento sino que indica el sentido de las llamadas a los métodos. Por ejemplo, en el modelo

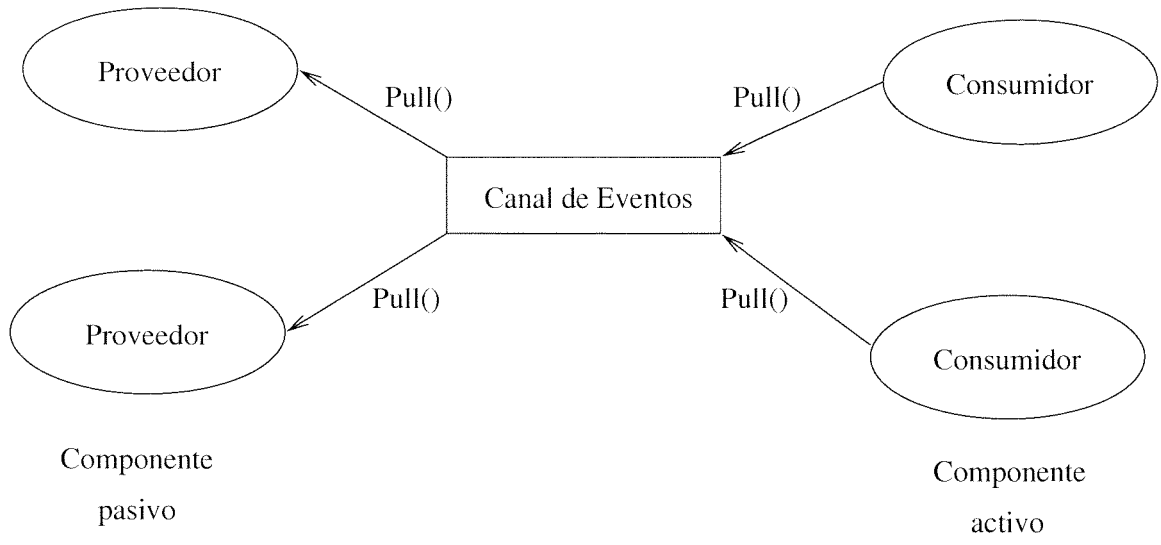


Figura 7.2: Modelo Pull

pull, el consumidor es el que realiza la llamada al método pull del canal y es el canal el que realiza la llamada al método pull del proveedor.

Aunque en las figuras mencionadas anteriormente sólo se consideran proveedores y consumidores que utilizan el mismo modelo, el canal puede gestionar la distribución de eventos entre objetos proveedores y consumidores que no utilizan el mismo modelo. De esta forma, un proveedor push puede enviar información de eventos a consumidores pull. El rol del canal de eventos varía dependiendo de la combinación de estos dos modelos.

El canal puede funcionar como un notificador de los eventos que envía el proveedor, como una cola donde poder almacenar la información de los eventos hasta que no son requeridas por el consumidor, como un procurador de eventos para el consumidor o como un agente activo que pide eventos al proveedor y se los envía al consumidor. Esta idea se esquematiza en la figura 7.4.

Conexión al Canal

Los proveedores se deben conectar al canal antes de proporcionar información. Por otra parte, los consumidores también se deben conectar al canal antes de consumir datos de eventos. La conexión a un canal es un proceso que se realiza en varios pasos.

Antes de detallar los pasos que hay que seguir para conectar un proveedor o un consumidor al canal, hay que introducir los distintos objetos que componen el canal y las políticas a llevar a cabo para la localización de éste.

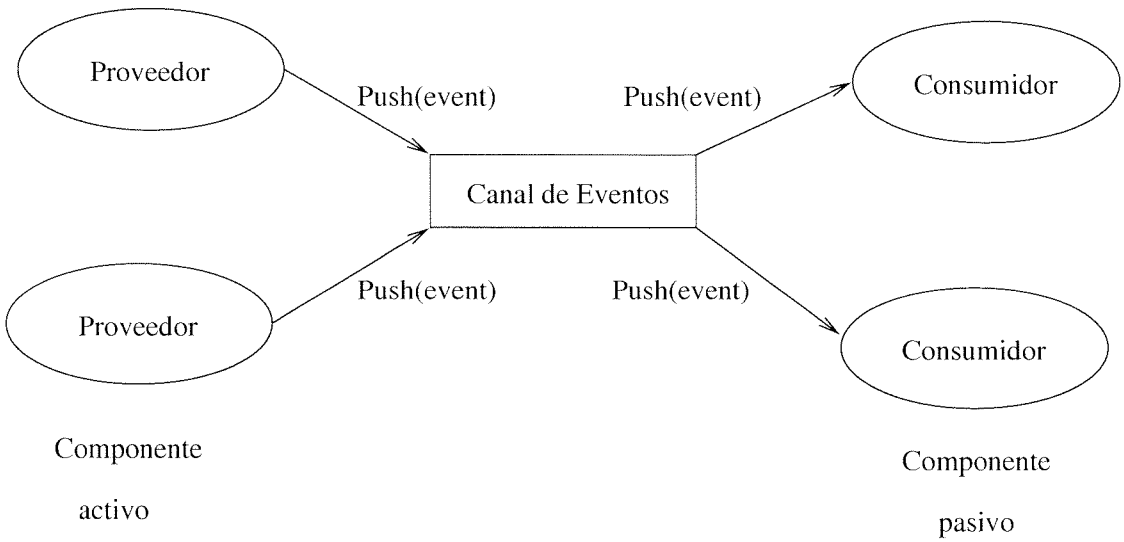


Figura 7.3: Modelo Push

El objeto Canal de Eventos es un agregado de un conjunto de objetos en los que delega ciertas responsabilidades. Las clases de estos objetos son las siguientes:

- Administrador de Consumidores (ConsumerAdmin): Se encarga de la gestión de los consumidores que se conectan al canal. A cada consumidor que se conecta le proporciona un objeto del tipo Proveedor Proxy.

- Administrador de Proveedores (SupplierAdmin): Se encarga de la gestión de los proveedores que se conectan al canal. A cada proveedor que se conecta le proporciona un objeto del tipo Consumidor Proxy.

- Proveedor Proxy del Modelo Push (ProxyPushSupplier): Proveedor perteneciente al canal que utiliza el modelo push de distribución de eventos y que actúa en nombre del proveedor real desde el punto de vista del consumidor.

- Proveedor Proxy del Modelo Pull (ProxyPullSupplier): Proveedor perteneciente al canal que utiliza el modelo pull de distribución de eventos y que actúa en nombre del proveedor real desde el punto de vista del consumidor.

- Consumidor Proxy del Modelo Push (ProxyPushConsumer): Consumidor perteneciente al canal que utiliza el modelo push de distribución de eventos y que actúa en nombre del consumidor real desde el punto de vista del proveedor.

- Consumidor Proxy del Modelo Pull (ProxyPullConsumer): Consumidor perteneciente al canal que utiliza el modelo pull de distribución de eventos y que actúa en nombre del consumidor real desde el punto de vista del proveedor.

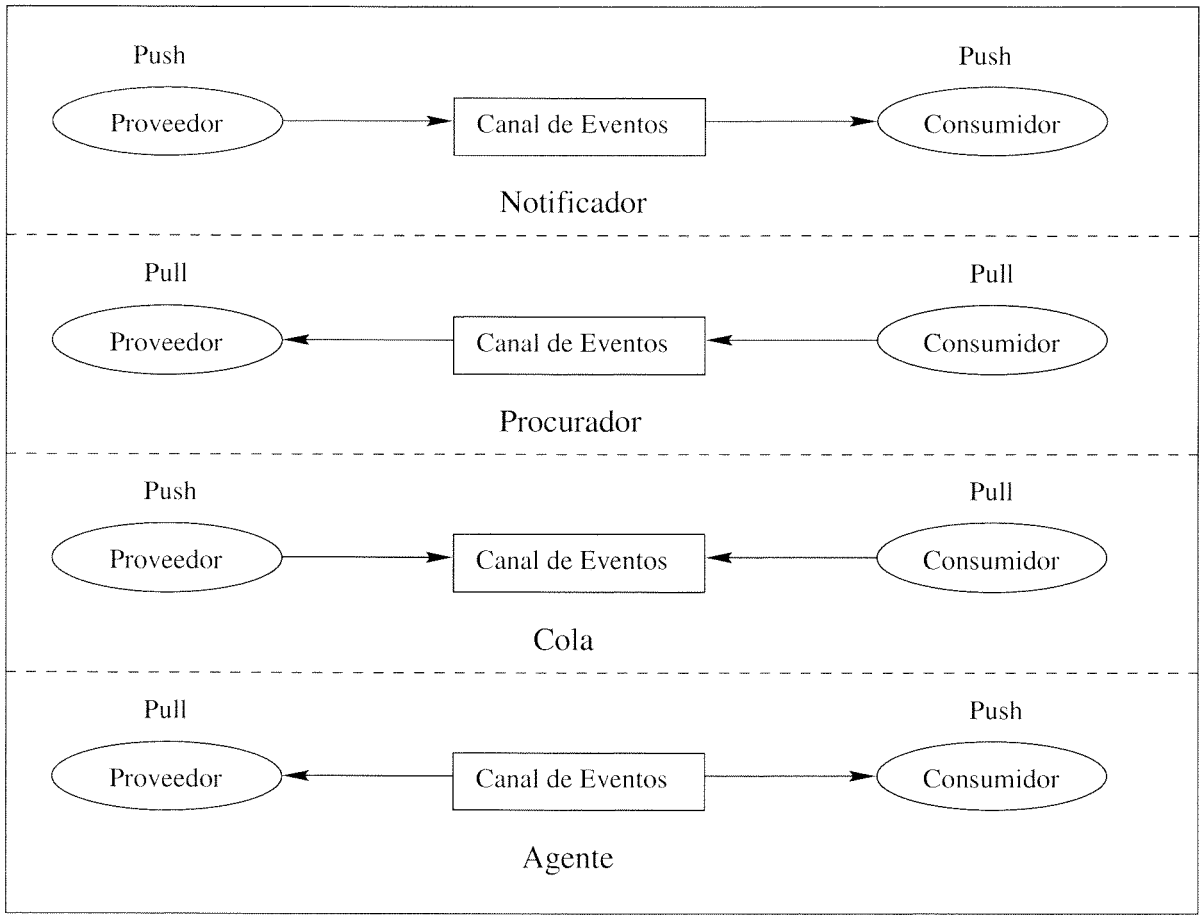


Figura 7.4: Rol del Canal de Eventos

Un canal de eventos estará compuesto por una instancia de la clase Administrador de consumidores, una instancia de la clase Administrador de proveedores y ninguna o varias instancias de cada una de las clases proxies.

Con respecto a las políticas para encontrar un canal, el servicio de eventos no establece una política concreta sino que deja esta responsabilidad a niveles más altos que dictarán:

- Cómo crear un canal.
- Cómo obtener una referencia al canal.

Por ejemplo el canal podría ser nombrado en un contexto de nombrado o bien podría ser exportado a través de una operación en otro objeto.

La conexión de un consumidor al canal se realiza siguiendo los pasos que se describen a continuación y que se pueden ver en la figura 7.5:

- Obtiene una referencia al objeto EventChannel requerido.
- Llama al método for_consumers del EventChannel para obtener una referencia al objeto ConsumerAdmin.
- Llama al método obtain_push_supplier del ConsumerAdmin para obtener una referencia al objeto ProxyPushSupplier.
- Llama al método connect_push_consumer del ProxyPushSupplier para conectar el consumidor al proveedor vía el canal de eventos.

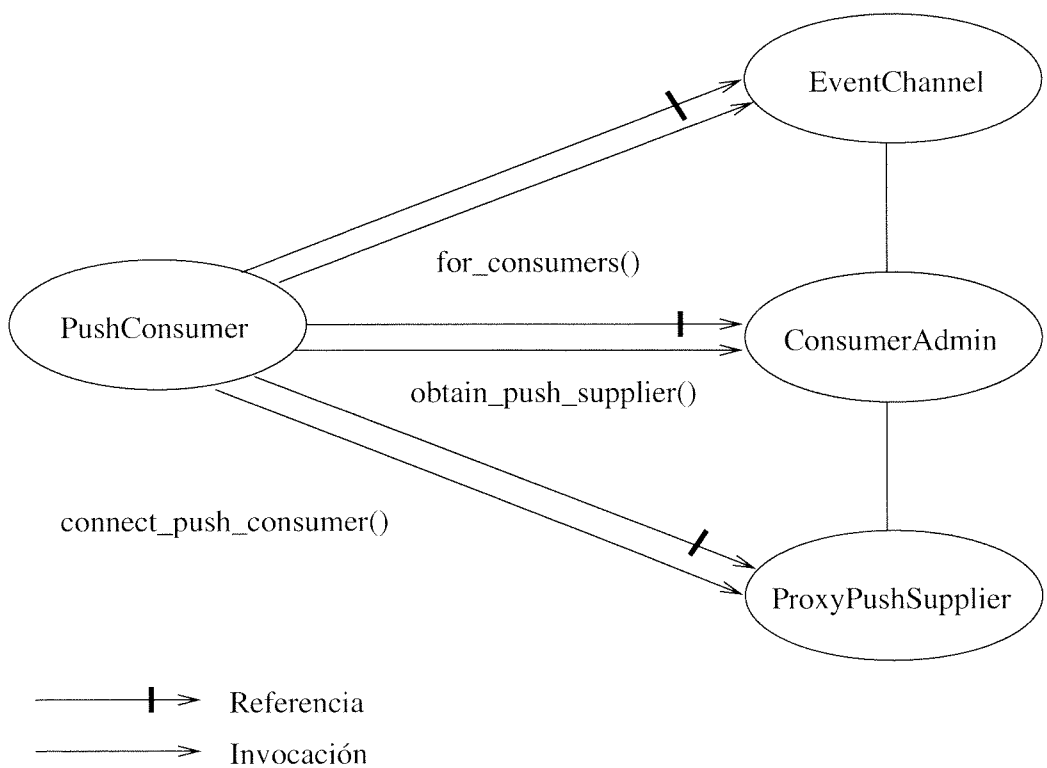


Figura 7.5: Conexión de un Consumidor

De una forma similar, la conexión de un proveedor se realiza siguiendo los pasos detallados a continuación y que se muestran en la figura 7.6 :

- Obtiene una referencia al objeto EventChannel requerido.
- Llama al método for_suppliers del EventChannel para obtener una referencia al objeto SupplierAdmin.

- Llama al método obtain_push_consumer del SupplierAdmin para obtener una referencia al objeto ProxyPushConsumer.

- Llama al método connect_push_supplier del ProxyPushConsumer para conectar el proveedor al consumidor vía el canal de eventos.

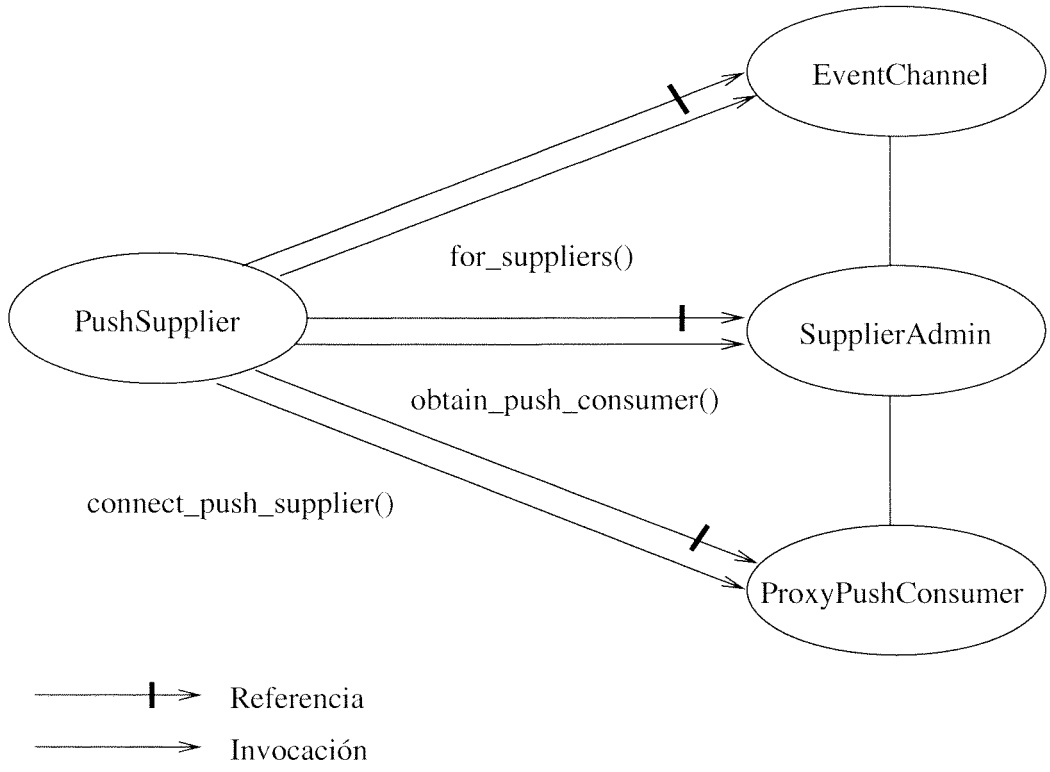


Figura 7.6: Conexión de un Proveedor

Los esquemas de conexión mostrados son los correspondientes al modelo Push, para el modelo Pull es similar y solamente hay que sustituir push por pull donde aparezca.

7.4.2 Elemento Proxy

En este segundo modelo utilizado para comunicar eventos producidos en el VMD a objetos CORBA, se ha introducido un elemento Proxy. Por una parte, este elemento comunica con el VMD y es capaz de recibir notificaciones de eventos que el VMD le envía. Por otra parte, este elemento comunica con el canal de eventos de CORBA, para difundir estas notificaciones a todos los objetos CORBA que hayan mostrado su interés. Este elemento es denominado MMSeventProxy y se comporta como un consumidor de eventos MMS y como un proveedor de eventos CORBA.

El modelo que mejor se ajusta a la naturaleza propia de la notificación de eventos y que se ha utilizado es el modelo Push. Sin embargo, en CORBA se podría haber utilizado también el modelo pull. Desde el punto de vista de los consumidores se puede utilizar cualquiera de los dos modelos aunque aquí se haya asumido el modelo push. Esto quiere decir, que MMSeventProxy manda las notificaciones de eventos al canal de eventos CORBA cuando son generadas por el VMD. El canal de eventos entonces, asume la responsabilidad de enviar estas notificaciones a los consumidores de estos eventos. Esta idea es esquematizada en la figura 7.7

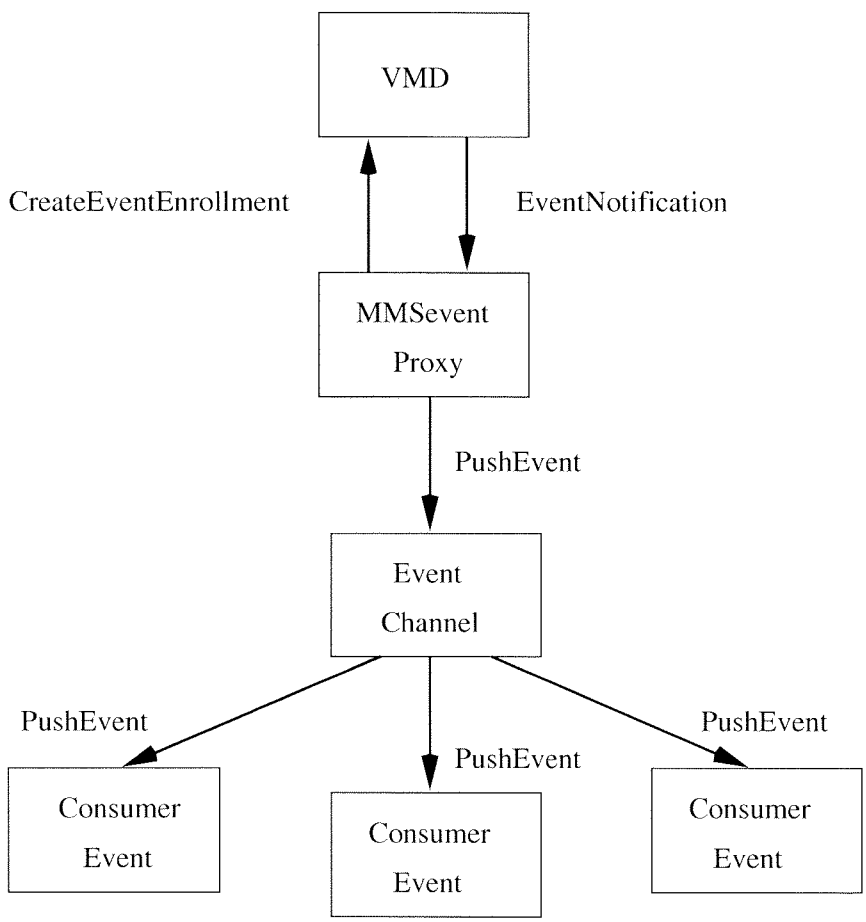


Figura 7.7: Comunicación de Eventos

El tipo de evento puede ser tipado o no tipado. Si el tipo de evento es tipado, el consumidor debe proporcionar la interfaz adecuada para la notificación del evento en base al tipo correspondiente al evento. Por otro lado, si el tipo de evento es no tipado, la notificación de evento se realiza comunicando un dato del tipo Any, aunque de todas formas necesite conocer el tipo real de los datos recibidos.

Las responsabilidades del objeto MMSeventProxy para permitir la comunicación de eventos a objetos CORBA son las siguientes:

- Creación del canal.
- Conexión al canal.
- Creación de registros de eventos en el VMD.
- Recepción de notificaciones.
- Transmisión de notificaciones.

En los siguientes apartados se detallan más detenidamente estas responsabilidades.

Creación del Canal

Crear el canal de eventos a través del cual los eventos van a ser transmitidos en el sistema CORBA. Una vez creado, el canal debe ser exportado de forma que los consumidores puedan encontrarlo en el bus de objetos. Para ello el elemento proxy utiliza el servicio de nombrado de CORBA, que permite obtener una referencia de objeto a través de un nombre.

Conexión al Canal

Indicar al canal que es un proveedor de eventos. El método adecuado en el canal para ser conectado debe ser llamado. Detalles de este procedimiento se han comentado en el apartado anterior. Como resultado, se obtiene una referencia al objeto ProxyPushConsumer. Por medio de esta referencia el objeto MMSeventProxy podrá enviar notificaciones de eventos al canal.

Creación de Registros de Eventos en el VMD

Crear los Registros de Evento en el VMD necesarios para ser notificado de los eventos MMS. Esto es llevado a cabo por medio del método Create Event Enrolment del servicio de eventos, donde se especifica la transición de la condición de evento que genera el evento. Además, el elemento Proxy debe ser exportado y debe indicar su nombre al VMD, de esta forma puede ser encontrado en el ORB usando el servicio de nombrado, en otro caso, le debe indicar su referencia. En este caso, MMSeventProxy actúa como un cliente del VMD.

Recepción de Notificaciones

Recibir las notificaciones de eventos que el VMD envía. El objeto MMSEventProxy debe estar preparado para ello, es decir, debe poder actuar como un servidor que procesa las notificaciones del VMD.

Transmisión de Notificaciones

Lanzar los eventos MMS recibidos usando el servicio de eventos de CORBA. Esto se lleva a cabo por medio del canal de eventos. Se lleva a cabo una petición Push Event al canal cuando la notificación MMS llega.

Capítulo 8

Conclusiones

En este capítulo final se presentan de forma resumida las contribuciones aportadas por este trabajo de tesis, las conclusiones a las que se han llegado, así como la vías futuras de investigación a las que pueden dar lugar los estudios realizados.

8.1 Contribuciones

Este trabajo es una aportación al uso de sistemas de gestión basados en CORBA en entornos de fabricación. Para ello, los puntos que se han tratado en los capítulos 3, 4 y 5 de esta tesis sirven de introducción al entorno en el que se ha desarrollado, y se pueden resumir en lo siguiente:

- Se ha descrito la arquitectura CORBA, profundizando en ella y viendo las posibilidades que tiene para el desarrollo de aplicaciones distribuidas.
- Se ha explorado en los sistemas de gestión actuales y la influencia que está teniendo CORBA en estos sistemas.
- Se ha dado una visión general del protocolo MMS, haciendo énfasis en su modelo de objetos.

Las conclusiones a las que se han llegado después de estos estudios son las siguientes:

- Se considera CORBA como la arquitectura para la utilización de objetos distribuidos más adecuada en sistemas de naturaleza heterogénea como son los sistemas

de fabricación.

- CORBA está teniendo una gran aceptación en la comunidad de gestión. Las arquitecturas tradicionales de gestión se están viendo recientemente influenciadas por las técnicas de objetos distribuidos, y aunque la adaptación de los sistemas actuales requiera un gran esfuerzo, todo apunta a una futura convivencia entre los métodos tradicionales de gestión y la utilización de objetos distribuidos.

- Se ha presentado MMS como un protocolo de comunicación apto como base para el desarrollo de una interfaz de objetos distribuidos que logra hacer uniforme el acceso a dispositivos de fabricación.

Las aportaciones de esta tesis se describen en los capítulos 6, 7 y 8, aunque hay que incluir también el apéndice C que pretende dar un carácter práctico a los modelos expuestos en estos capítulos.

Estas contribuciones se pueden resumir en las siguientes:

- Se ha planteado la estructura general de una plataforma de gestión basada en CORBA, comentando cada uno de los componentes de la arquitectura.

- Se ha estudiado la forma de contemplar los dispositivos de fabricación desde el punto de vista de la gestión, integrándolos en un sistema de gestión basado en CORBA.

- Se han presentado los distintos aspectos necesarios para la adaptación del protocolo MMS, tanto el modelo de comunicación, como los servicios y las unidades de datos del protocolo, usando la especificación del JIDM para la traducción de ASN.1 a IDL.

- Se han propuesto dos modelos diferentes para lograr la comunicación de eventos producidos en el dispositivo de fabricación. El primero siguiendo el modelo MMS y el segundo utilizando el servicio de eventos proporcionado por CORBA.

- Se ha desarrollado un prototipo en Java para la gestión de sistemas de fabricación siguiendo las bases expuestas en este trabajo. Como funcionalidad adicional se ha incluido un componente que permite la ejecución de aplicaciones distribuidas teniendo en cuenta la concurrencia y sincronización entre tareas.

Los apéndices A y B son un resumen de IDL y de ASN.1 respectivamente ya que han sido muy utilizados en el desarrollo de este trabajo de tesis.

8.2 Conclusiones

Como conclusiones una vez realizado este trabajo se puede indicar que:

- La arquitectura CORBA sigue evolucionando y mejorando tanto la arquitectura en sí como los servicios y facilidades que ofrece, por lo que se siguen abriendo caminos para la mejora de las aplicaciones distribuidas.

- Los sistemas de gestión han sido influenciados por esta arquitectura consiguiendo hacer más flexibles estos sistemas que ofrecen posibilidades muy importantes y necesarias en el adecuado seguimiento y correcto funcionamiento de los sistemas distribuidos.

- Debido a lo expuesto en el punto anterior, los sistemas de gestión se están ampliando no sólo para gestionar dispositivos de red como en un principio, sino que se está contemplando la integración de todos los dispositivos que están implicados en aplicaciones distribuidas así como la propia aplicación en estos sistemas de gestión.

- Se considera MMS como un protocolo que ofrece muchas posibilidades a la hora de monitorizar y controlar dispositivos de fabricación, ya que es un protocolo que ha surgido en estos entornos y posibilita tanto la ejecución de programas como el acceso a variables y la comunicación de eventos.

- Con la introducción de un objeto CORBA que presenta una interfaz a los dispositivos de fabricación, se logra abrir una vía para la gestión de estos dispositivos usando el ORB como el mecanismo de comunicación.

- La adaptación del protocolo MMS especificado en ASN.1 a IDL se realiza en base a la especificación del JIDM, necesitando un mínimo de modificaciones para el desarrollo de la interfaz al Dispositivo de Fabricación Virtual. Debido a que esta adaptación se ajusta en gran medida al protocolo, futuras implementaciones de puentes entre el sistema CORBA y otras formas de implementar MMS resultarán en gran medida muy intuitivas.

- Tanto el modelo de eventos de MMS como el servicio de eventos de CORBA pueden ser utilizados para realizar las notificaciones de eventos que se producen en un sistema de fabricación. Dependiendo de la naturaleza de la aplicación distribuida será más conveniente utilizar o no el servicio de eventos de CORBA.

8.3 Líneas Futuras de Investigación

Como líneas de continuación de la investigación llevada a cabo en este trabajo y directamente relacionadas con éste se podrían señalar entre otras las siguientes:

- Añadir nuevas funcionalidades al gestor adicionales a las ya comentadas. Por ejemplo se podría considerar el sondeo de variables o registro de los eventos producidos.
- Continuar con la adaptación del protocolo MMS, ya que hay parte del protocolo que no se ha contemplado, aunque sí se han contemplado los objetos más importantes y utilizados.
- Aplicación de ORB's de tiempo real que hagan uso de un servicio de eventos de tiempo real [Har97] a los estudios realizados.
- Implementación de los objetos integradores que permitan comunicar con dispositivos que ya utilicen MMS bien sobre OSI o bien sobre TCP/IP.

Como líneas de trabajo futuro que también están relacionadas con éste pero no tan directamente se pueden mencionar entre otras:

- Identificar otros campos en los que se podrían aplicar estas propuestas, por ejemplo en teleoperación o en control.
- Estudio de la integración con otros tipos de gestión no basados en CORBA.

Apéndice A

OMG IDL

En este anexo se describe el Lenguaje de Definición de Interfaz (IDL) de OMG. Es el lenguaje usado para describir las interfaces proporcionadas por las implementaciones de los objetos y que pueden ser llamadas por los objetos clientes. Una interfaz IDL proporciona toda la información que se necesita para el desarrollo de clientes que usan las operaciones de esta interfaz.

IDL es un lenguaje puramente declarativo, es decir, no se utiliza para la implementación de los objetos, pero se ha definido un conjunto de mapeados de IDL a distintos lenguajes de programación. El mapeado depende de las facilidades que proporcione el lenguaje al que se va a mapear.

A lo largo de este anexo se van a comentar las convenciones léxicas, el preprocesamiento y la estructura de una especificación IDL, la herencia, la declaración de constantes, tipos, excepciones, operaciones y atributos, el nombrado y ámbito de los nombres, así como las diferencias entre IDL y C++. La descripción de la gramática IDL utiliza una notación sintáctica similar al formato extendido de Backus-Naur (EBNF, Extended Backus-Naur Format). Los símbolos utilizados se pueden ver en la tabla A.1.

Se puede consultar la gramática completa del lenguaje IDL en la especificación CORBA [Obj95b].

A.1 Convenciones Léxicas

Se trata en este apartado de los comentarios, identificadores y palabras clave en IDL.

Símbolo	Significado
::=	Es definido como
|	Alternativa
<texto>	No terminal
"text"	Literal
*	La unidad sintáctica precedente puede ser repetida cero o más veces
+	La unidad sintáctica precedente puede ser repetida una o más veces
{}	Encierra unidades sintácticas que pueden ser agrupadas como una única unidad sintáctica
[]	La unidad sintáctica que encierra es opcional (puede ocurrir cero o una vez)

Tabla A.1: Notación EBNF de IDL

Comentarios

Los comentarios en una especificación IDL se añaden de la misma forma que en C++, es decir, los caracteres /* comienzan un comentario que finaliza con */ y los caracteres // comienzan un comentario que finaliza cuando termina la línea.

Identificadores

Un identificador es una secuencia de longitud arbitraria de caracteres alfabéticos, dígitos y caracteres de subrayado. El primer carácter debe ser un carácter alfabético. En IDL no se hace distinción entre las letras mayúsculas y minúsculas, es decir, si hay dos identificadores que difieren sólo en las mayúsculas y minúsculas se considera redefinición. Pero por otro lado, todas las referencias a una definición deben ser idénticas. Esto es así para facilitar el mapeado a lenguajes que son "case insensitive" y a lenguajes que son "case sensitive".

Sólo hay un espacio de nombres para los identificadores IDL, esto quiere decir que no se puede utilizar el mismo identificador para una constante y para una interfaz, ya que esto daría problemas de compilación.

Palabras Claves

Las palabras claves para IDL se muestran en la tabla B.2.

any	double	interface	readonly	unsigned
attribute	enum	long	sequence	union
boolean	exception	module	short	void
case	FALSE	Object	string	wchar
char	fixed	octet	struct	wstring
const	float	oneway	switch	
context	in	out	TRUE	
default	inout	raises	typedef	

Tabla A.2: Palabras claves de IDL

Estas palabras son reservadas y no se pueden utilizar por ejemplo como identificadores, deben ser escritas exactamente como se muestra en la tabla, cambios en mayúsculas o minúsculas darán problemas de compilación.

A.2 Preprocesamiento

En un fichero de definición se pueden incluir directivas para el preprocesador. El preprocesamiento en IDL está basado en el de ANSI C++. Se pueden incluir directivas para la sustitución de macros, compilación condicional e inclusión de ficheros fuentes.

A.3 Especificación IDL

Una especificación IDL consiste de una o más definiciones de tipo (<type_dcl>) , definiciones de constantes (<const_dcl>), definiciones de excepciones (<except_dcl>), definiciones de interfaces (<interface>) o definiciones de módulos (<module>). La sintaxis es la siguiente:

<specification> ::= *<definition>+*

<definition> ::= *<type_dcl>* ";"
| *<const_dcl>* ";"
| *<except_dcl>* ";"
| *<interface>* ";"
| *<module>* ";"

A.4 Declaración de Módulos

La declaración de un módulo se realiza con la palabra clave "module" seguida de un identificador que corresponde al nombre de dicho módulo y seguida de una o más definiciones entre llaves. Las definiciones son las vistas anteriormente.

La sintaxis de la declaración de módulos es la siguiente:

<module> ::= *"module"* *<identifier>* *"{"* *<definition>+* *"}"*

A.5 Declaración de Interfaces

La declaración de una interfaz incluye una cabecera y un cuerpo de la interfaz.

La cabecera se declara indicando la palabra clave "interfaz" seguida de un identificador que es el nombre de la interfaz y opcionalmente se puede añadir ":" y especificar un conjunto de interfaces definidas previamente de las cuales hereda la interfaz que se está especificando separadas por ",".

El cuerpo de la interfaz incluye declaraciones de constantes, de tipos, de excepciones, de atributos y operaciones que se comentan en los apartados siguientes.

Son permitidas las interfaces vacías, que no contienen ninguna declaración. Se pueden también declarar las interfaces sin especificar la herencia ni el cuerpo, para utilizarlas antes incluso de completar la declaración. En este caso, solamente se indica el nombre de la interfaz.

La definición de la interfaz satisface la siguiente sintaxis:

```
<interface>          ::= <interface_dcl>
                       |  <forward_dcl>

<interface_dcl>      ::= <interface_header> "{" <interface_body> "}"

<forward_dcl>        ::= "interface" <identifier>

<interface_header>   ::= "interface" <identifier> "[" <inheritance_spec> "]"

<interface_body>     ::= <export>*

<export>             ::= <type_dcl> ";"
                       |  <const_dcl> ";"
                       |  <except_dcl> ";"
                       |  <attr_dcl> ";"
                       |  <op_dcl> ";"

<inheritance_spec>   ::= ":" <scoped_name> { "," <scoped_name> }*

<scoped_name>        ::= <identifier>
                       |  "::" <identifier>
                       |  <scopec_name> "::" <identifier>
```

A.6 Herencia

Una interfaz puede derivar de otra interfaz, a la primera se le conoce como interfaz derivada y a la segunda como interfaz base. Los elementos definidos en la interfaz base son entonces heredados por la interfaz derivada, y pueden ser utilizados en ésta. La interfaz derivada puede redefinir nombres de la clase base, y para hacer referencia a los de la base será necesario utilizar el operador de resolución de nombres "::".

En IDL está permitida la herencia múltiple, es decir, un interfaz puede derivar a la vez de varios interfaces. Hay que tener en cuenta que la referencia a los elementos de una interfaz base no debe ser ambigua. La referencia es ambigua cuando se utiliza un nombre que se encuentra en más de una interfaz base. En este caso hay que incluir en la referencia el nombre de la interfaz a la que pertenece para eliminar la ambigüedad.

A.7 Declaración de Constantes

En la declaración de una constante se indica la palabra clave "const" seguida del tipo de la constante, el nombre, el signo "=" y una expresión constante. La sintaxis de una declaración de constantes es la siguiente:

<const_dcl> ::= *"const" <const_type> <identifier> "=" <const_exp>*

A.8 Declaración de Tipos

Las declaraciones de tipos son declaraciones que asocian un identificador a un tipo. Se utiliza como en C la palabra clave "typedef" para asociar un nombre con un tipo, aunque también la declaración de estructuras, uniones, enumerados y declaraciones nativas asocian un nombre con un tipo de dato.

Los distintos tipos que se pueden declarar son los tipos básicos, los tipos construidos, los tipos patrones y las matrices.

Tipos Básicos

Los tipos básicos son los que se muestran en la tabla B.3.

Tipos Construidos

Los tipos construidos son los siguientes:

- Estructuras: Una estructura indica un conjunto relacionado de elementos de distinto tipo. La sintaxis de la declaración de una estructura es la siguiente:

  <struct_type> ::= *"struct" <identifier> "{" <member_list> "}"*

- Uniones discriminadas: Una unión discriminada contendrá un elemento de un tipo u otro dependiendo del valor del discriminante. En la cabecera de la unión se declara el discriminante que puede ser de tipo entero, carácter, booleano, enumerado o de un tipo definido que sea del tipo de alguno de los anteriores. En el cuerpo de la unión discriminada se tiene la declaración de elementos de

Tipo entero	short	$-2^{15}..2^{15}-1$
	long	$-2^{31}..2^{31}-1$
	long long	$-2^{63}..2^{63}-1$
	unsigned short	$0..2^{16}-1$
	unsigned long	$0..2^{32}-1$
	unsigned long long	$0..2^{64}-1$
Tipo flotante	float	Simple precisión
	double	Doble precisión
	long double	Extendido doble
Tipo carácter	char	Carácter de 8 bits
	wchar	Carácter extendido
Tipo booleano	boolean	Verdadero o falso
Tipo octeto	octet	8 bits
Tipo any	any	Cualquier tipo IDL

Tabla A.3: Tipos básicos de IDL

distinto tipo según distintos valores del discriminante, de una forma similar a una sentencia "switch" de C, es decir, en distintas sentencias "case". El cuerpo también puede incluir una sentencia "default", que indica el contenido de la unión en el caso de que el discriminante no tenga ninguno de los valores indicados en las distintas sentencias "case". No es necesario que en el cuerpo de la unión se listen todos los valores posibles del discriminante. La sintaxis de la declaración es la siguiente:

<union_type> ::= *"union" <identifier> "switch"*
"(" <switch_type_spec> ")"
"{" <switch_body> "}"

<switch_type_spec> ::= *<integer_type>*
| *<char_type>*
| *<boolean_type>*
| *<enum_type>*
| *<scoped_name>*

<switch_body> ::= *<case>+*

<case> ::= *<case_label>+ <element_spec> ";"*

<case_label> ::= *"case" <const_exp> ":"*
| *"default" ":"*

<element_spec> ::= *<type_spec> <declarator>*

- Enumeraciones: Un tipo enumerado consiste de una lista ordenada de identificadores. El orden en el que los identificadores son nombrados en la especificación definen el orden relativo de los identificadores. La sintaxis de la enumeración es la siguiente:

  <enum_type> ::= *"enum" <identifier>*
  "{" <enumerator> { "," <enumerator> } "}"*

  <enumerator> ::= *<identifier>*

Tipos Patrones

Se consideran tipos patrones los tipos siguientes:

- Secuencia (sequence): una secuencia es un array unidimensional que tiene un tamaño máximo fijado en tiempo de compilación y una longitud determinada en tiempo de ejecución. La sintaxis del tipo secuencia es el siguiente:

  <sequence_type> ::= *"sequence"*
  "<" <simple_type_spec> "," <positive_int_const> ">"
  | *"sequence" "<" <simple_type_spec> ">"*

- Tipo cadena (string): Es similar a una secuencia de caracteres (char). Se puede indicar por tanto un tamaño máximo para la cadena. La sintaxis es la siguiente:

  <string_type> ::= *"string" "<" <positive_int_const> ">"*
  | *"string"*

- Tipo cadena extendida (wstring): Es similar a una cadena, pero en este caso representa una secuencia de caracteres extendidos (wchar).

- Tipo de punto fijo (fixed): Representa un número decimal de punto fijo de hasta 31 dígitos significativos.

Matrices

Se pueden definir arrays multidimensionales de tamaño fijo, indicando el tamaño para cada dimensión. La sintaxis es la siguiente:

<array_declarator> ::= *<identifier> <fixed_array_size>+*

<fixed_array_size> ::= *"[" <positive_int_const> "]"*

A.9 Declaración de Excepciones

Una excepción indica un condición excepcional que ha ocurrido durante la realización de una petición. El valor del identificador de la excepción es accesible al programador permitiendo la determinación de la excepción particular. Si se declaran miembros en la excepción, cuando ocurre se puede consultar esta información adicional. La sintaxis de la declaración de una excepción es la siguiente:

<except_dcl> ::= *"exception"* *<identifier>* *"{"* *<member>** *"}"*

Existen un conjunto de excepciones estándares ya definidas que pueden ser generadas por el ORB, como por ejemplo excepción de objeto no existente o fallo de comunicación. El listado de todas las excepciones estándares se puede consultar en la especificación CORBA [Obj95b].

A.10 Declaración de Operaciones

La declaración de una operación en IDL es similar a la declaración de funciones en C. La declaración consiste de :

- Atributo de operación: Es opcional e indica la semántica que proporciona el servicio de comunicación para las invocaciones de una operación. Se puede especificar el atributo "oneway" indicando que la semántica de la invocación es "mejor esfuerzo" que no garantiza la llamada ya que implica que la invocación será realizada una vez como máximo. Una operación con este atributo no debe tener ningún parámetro de salida, el tipo de retorno debe ser void y no puede incluir ninguna expresión de disparo de excepción. Si no se especifica este atributo la semántica que se aplica es la de "como máximo una vez" si ocurre una excepción o "exactamente una vez" si la invocación se lleva a cabo con éxito.

- Tipo del resultado de la operación: Puede ser cualquier tipo que pueda ser especificado en IDL. Se especifica "void" si la operación no devuelve nada.

- Identificador de la operación: Nombre de la operación en el ámbito de la interfaz en la que está definida.

- Lista de parámetros: Es una lista de 0 o más declaraciones de parámetros. La declaración de un parámetro incluye un atributo direccional que indica si es de entrada (in), salida (out), o entrada/salida (inout), un tipo y un nombre.

- Expresión de disparo de excepción (raises): Indica las excepciones que pueden ocurrir como resultado de la invocación de la operación. Es opcional. Todas las excepciones nombradas en la declaración de la operación deben ser previamente definidas.
- Expresión de contexto (context): Es opcional y especifica elementos del contexto del cliente que pueden afectar al rendimiento de una petición.

La sintaxis de la declaración de operación es la siguiente:

```
<op_dcl>            ::= [ <op_atribute> ] <op_type_spec> <identifier>
                        <parameter_dcls> [ <raises_expr> ] [ <context_expr> ]

<op_type_spec>      ::= <param_type_spec>
                      | "void"

<op_dcl>            ::= "oneway"

<parameter_dcls>    ::= "(" <param_dcl> { "," <param_dcl> }* ")"
                      | "(" ")"

<param_dcl>         ::= <param_attribute> <param_type_spec> <simple_declarator>

<param_attribute>   ::= "in"
                      | "out"
                      | "inout"

<param_type_spec>   ::= <base_type_spec>
                      | <string_type>
                      | <scoped_name>

<raises_expr>       ::= "raises" "(" <scoped_name> { "," <scoped_name> }* ")"

<context_expr>      ::= "context" "(" <string_literal> { "," <string_literal> }* ")"
```

A.11 Declaración de Atributos

En una interfaz se pueden declarar atributos además de operaciones. Declarar un atributo es equivalente a declarar dos funciones de acceso al atributo, una para obtener el valor y otra para actualizarlo. Si se declara el atributo de solo lectura, entonces sólo

se tiene la función de recuperación del valor, no la de actualización. La sintaxis de la declaración de atributo es la siguiente:

<attr_dcl> ::= [*"readonly"*] *"attribute"* *<param_type_spec>*
<simple_declarator> { "," *<simple_declarator>* }*

A.12 Nombres y Ámbito

Los nombres definidos en un ámbito no pueden ser redefinidos es ese mismo ámbito. Un fichero IDL forma un ámbito de nombrado. Adicionalmente los tipos siguientes forman ámbitos anidados:

- Módulo
- Interfaz
- Estructura
- Unión
- Operación
- Excepción

A.13 Diferencias con C++

Aunque la gramática de IDL es muy similar a la de C++, existen las siguientes restricciones:

- El valor resultado de una operación es obligatorio.
- En la declaración de una operación, se debe proporcionar el nombre de cada parámetro formal.
- No se permite indicar "void" en la lista de parámetros de una operación para indicar lista vacía.
- Las estructuras, uniones discriminadas y enumeraciones deben tener una etiqueta que identifique el tipo.

- Los enteros no pueden ser definidos simplemente como int o unsigned, deben ser declarados explícitamente como short o long.
- Un carácter no puede ser definido como signed o unsigned.

Apéndice B

Notación de Sintaxis Abstracta Uno (ASN.1)

La Notación de Sintaxis Abstracta Uno (ASN.1 Abstract Sintax Notation One) definida en la recomendación X.680 del UIT-T (ISO/CEI 8824-1) [Rec94a] proporciona un método para la especificación de tipos de datos utilizados en protocolos de comunicación. Se considera abstracta ya que no determina la forma de representar una instancia de un tipo definido con el fin de realizar la transferencia. La representación de los datos se trata en la recomendación X.690 del UIT-T (ISO/CEI 8825-1) [Rec94b] donde se define una sintaxis de transferencia. Esta forma de representar la información sirve para transmitir las instancias de los tipos de datos definidos utilizando ASN.1.

La técnica utilizada para la definición de un tipo complejo en ASN.1 consiste en la definición de un conjunto reducido de tipos simples y combinar estos tipos simples para obtener tipos estructurados. En este anexo se presenta de forma resumida algunas convenciones léxicas de la notación, la definición de módulos, los distintos tipos simples y los tipos estructurados. No se tratará de la sintaxis de transferencia ya que no es determinante para el trabajo realizado.

Para clarificar la explicación se ha incluido parte de la gramática de ASN.1, para consultar la gramática completa, ésta se puede obtener en la recomendación mencionada anteriormente. En este caso la gramática utiliza una notación sintáctica similar al formato extendido de Backus-Naur (EBNF, Extended Backus-Naur Format), y los símbolos utilizados se muestran en la tabla B.1.

Símbolo	Significado
::=	Es definido como
|	Alternativa
SimboloNoTerminal	No terminal (aunque algunos símbolos terminales tienen esta forma)
simboloterminal	Terminal
SIMBOLOTERMINAL	Terminal
"Literal"	Literal
*	La unidad sintáctica precedente puede ser repetida cero o más veces
+	La unidad sintáctica precedente puede ser repetida una o más veces

Tabla B.1: Notación EBNF de ASN.1

B.1 Convenciones Léxicas

B.1.1 Comentarios

Los comentarios en una definición ASN.1 se puede incluir comenzando con un par de guiones adyacentes y terminando al final de la línea o con otro par de guiones adyacentes.

B.1.2 Identificadores, Referencias de Tipos y Nombres de Módulos

Los identificadores, las referencias de tipo y los nombres de módulos estarán constituido por un número arbitrario de una o más letras, dígitos o guiones. Un guión no será el último carácter. Un guión no irá seguido inmediatamente de otro guión. El carácter inicial será una letra minúscula para los identificadores, y mayúscula para las referencias de tipos y nombres de módulos.

B.1.3 Palabras Reservadas

Las palabras reservadas se muestran en la tabla B.2.

ABSENT	EMBEDDED	INSTANCE	REAL
ABSTRACT-SYNTAX	END	INTEGER	SEQUENCE
ALL	ENUMERATED	INTERSECTION	SET
APPLICATION	EXCEPT	ISO646String	SIZE
AUTOMATIC	EXPLICIT	MAX	STRING
BEGIN	EXPORTS	MIN	SYNTAX
BIT	EXTERNAL	MINUS-INFINITY	T61String
BMPString	FALSE	NULL	TAGS
BOOLEAN	FROM	NumericString	TeletexString
BY	GeneralizedTime	OBJECT	TRUE
CHARACTER	GeneralString	ObjectDescriptor	UNION
CHOICE	GraphicString	OCTET	UNIQUE
CLASS	IA5String	OF	UNIVERSAL
COMPONENT	TYPE-IDENTIFIER	OPTIONAL	UniversalString
COMPONENTS	IDENTIFIER	PDV	UTCTime
CONSTRAINED	IMPLICIT	PLUS-INFINITY	VideotexString
DEFAULT	IMPORTS	PRESENT	VisibleString
DEFINITIONS	INCLUDES	PrintableString	WITH
		PRIVATE	

Tabla B.2: Palabras reservadas en ASN.1

B.2 Definición de Módulo

Un módulo es el bloque de construcción básico de una especificación ASN.1. Un módulo se compone de los siguientes elementos:

- Identificador de módulo: Es el nombre del módulo seguido opcionalmente por un identificador de objeto que identifica al módulo.
- Cuerpo del módulo: A su vez está constituido por:
  - Símbolos exportados: Lista de símbolos definidos en el módulo y que pueden ser referenciados desde el exterior del módulo.
  - Símbolos importados: Lista de símbolos que son utilizados en el módulo pero que han sido definidos en otros módulos y por tanto deben ser importados de éstos.
  - Lista de asignaciones: Contiene un conjunto de definiciones de tipo, definiciones de valor y definiciones de macro. Una definición de tipo permite declarar un nuevo tipo de dato. Una definición de valor se usa para indicar el valor de un objeto de un tipo definido. Las definiciones de macros per-

miten al usuario extender la sintaxis de ASN.1 para definir nuevos tipos y sus valores.

La sintaxis de la definición de un módulo es la siguiente:

ModuleDefinition ::= *ModuleIdentifier DEFINITIONS "::="*
BEGIN
ModuleBody
END

ModuleBody ::= *Exports Imports AssignmentList*

Exports ::= *EXPORTS SymbolsExported ";"*
| *empty*

Imports ::= *IMPORTS SymbolsImported ";"*
| *empty*

SymbolsImported ::= *SymbolsFromModuleList*
| *empty*

SymbolsFromModuleList ::= *SymbolsFromModule*
| *SymbolsFromModuleList SymbolsFromModule*

SymbolsFromModule ::= *SymbolList FROM GlobalModuleReference*

AssignmentList ::= *Assignment*
| *AssignmentList Assignment*

Assignment ::= *TypeAssignment*
| *ValueAssignment*

TypeAssigment ::= *typereference "::=" Type*

ValueAssignment ::= *valuereference Type "::=" ValueSet*

B.3 Definición de Tipos

Una definición de tipo se realiza como se ha podido ver en el apartado anterior en la producción *TypeAssigment* mediante la asignación de un tipo a una referencia de tipo. Los tipos pueden ser bien tipos básicos o tipos estructurados.

En ASN.1 cada tipo de dato (con excepción de la elección y el tipo ANY) tienen asociado una etiqueta o rótulo que los identifica. Una etiqueta o rótulo consiste en un nombre de clase y un número de etiqueta. Hay cuatro clases de etiquetas:

- UNIVERSAL (universal): tipos independientes de la aplicación, definidos en el estándar.
- APPLICATION (aplicación): tipos que son de interés para la aplicación.
- Context-specific (específica del contexto): tipos de interés para la aplicación pero aplicables únicamente en un contexto limitado.
- Private (privada): tipos definidos por el usuario.

El número de etiqueta es un número entero no negativo.

Se pueden definir tipos rotulados que son tipos nuevos derivados de otros tipos ya existentes. El tipo rotulado se utiliza principalmente donde se requiera el empleo de tipos con etiquetas distintas para hacer posible que el receptor interprete la información correctamente. La rotulación puede ser implícita (IMPLICIT) o explícita (EXPLICIT). La rotulación implícita indica que la etiqueta del tipo rotulado no es necesaria en la transferencia, se consigue de esta forma una transferencia de octetos mínima.

B.3.1 Tipos Básicos

Los tipos básicos son tipos atómicos que no contienen componentes. Los tipos básicos en ASN.1 son los mostrados en la tabla B.3.

B.3.2 Tipos Estructurados

Un tipo estructurado es un tipo compuesto de otros tipos. Los tipos estructurados son los que se comentan en los siguientes subapartados.

B.3.2.1 Tipo Secuencia

El tipo secuencia se utiliza para denotar una lista ordenada de valores de distintos tipos. Se puede indicar que un componente es opcional (OPTIONAL), es decir, que

Tipo	Notación	Significado
Booleano	BOOLEAN	Verdadero o falso
Entero	INTEGER	Números enteros tanto positivos como negativos
Enumerado	ENUMERATED	Lista de valores enteros que puede tomar una instancia de este tipo
Real	REAL	Números reales
Cadena de bits	BIT STRING	Secuencia de 0 o más bits
Cadena de octetos	OCTET STRING	Secuencia de 0 o más octetos
Nulo	NULL	Valor nulo
Identificador de objeto	OBJECT IDENTIFIER	Conjunto de valores asociados con objetos de información
Cadenas de caracteres	BMPString GeneralString GraphicString IA5String ISO646String NumericString PrintableString TeletexString T61String UniversalString VideoString VisibleString	Cadena de caracteres procedentes de algún conjunto de caracteres especificado (dígitos decimales, imprimibles, ASCII, etc)
Externo	EXTERNAL	Tipo no especificado en ASN.1
Tiempo	UTCTime GeneralizedTime	Fecha expresada en año, mes, día y la hora expresada en hora, minutos y segundos

Tabla B.3: Tipos básicos de ASN.1

no es necesario que esté presente en la secuencia. Se puede indicar también un valor por defecto (DEFAULT), es decir, en el caso de que el componente no esté presente en la secuencia, se tomará este valor por defecto. Se pueden incluir en la secuencia los componentes de (COMPONENTS OF) otro tipo referenciado. La sintaxis es la siguiente:

SequenceType ::= *SEQUENCE* "{" *ComponentTypeList* "}"
| *SEQUENCE* "{" "}"

ComponentTypeList ::= *ComponentType*
| *ComponentTypeList* "," *ComponentType*

ComponentType ::= *NamedType*
| *NamedType OPTIONAL*
| *NamedType DEFAULT Value*
| *COMPONENTS OF Type*

B.3.2.2 Tipo "Secuencia de"

El tipo "secuencia de" se utiliza para denotar una lista ordenada de valores del mismo tipo. La sintaxis es la siguiente:

SequenceOfType ::= *SEQUENCE OF Type*

B.3.2.3 Tipo Conjunto

El tipo conjunto se utiliza para denotar una lista de valores de diversos tipos, donde el orden no es significativo. La sintaxis es la siguiente:

SetType ::= *SET* "{" *ComponentTypeList* "}"
| *SET* "{" "}"

B.3.2.4 Tipo "Conjunto de"

El tipo "conjunto de" se utiliza para denotar una lista de valores del mismo tipo, donde el orden no es significativo. La sintaxis es la siguiente:

SetOfType ::= *SET OF Type*

B.3.2.5 Tipo Elección

Un tipo elección es un lista de alternativas de tipos conocidos. El valor de una instancia de un tipo elección será un valor de uno de estos tipos. La sintaxis es la siguiente:

$$\begin{aligned} \textit{ChoiceType} &::= \textit{CHOICE } \text{“}\{\text{”} \textit{ AlternativeTypeList } \text{“}\}\text{”} \\ \textit{AlternativeTypeList} &::= \textit{NamedType} \\ &\quad \mid \textit{AlternativeTypeList } \text{“,”} \textit{ NamedType} \end{aligned}$$

Apéndice C

Prototipo en Java

Como parte de este trabajo de tesis se ha construido un prototipo [Ari98b] que permite la gestión de los dispositivos de fabricación de forma remota, utilizando CORBA como la plataforma para la comunicación de los distintos elementos del sistema y la interfaz MMS adaptada a IDL para realizar las distintas operaciones en estos dispositivos.

En este anexo se presenta el prototipo realizado. Para ello se expone la funcionalidad, se comenta la plataforma de implementación concreta que se ha elegido, tanto software como hardware, se muestra el modelo de objetos de la aplicación, tanto del gestor como del objeto gestionado (servidor MMS), se resume el procedimiento de desarrollo llevado a cabo y se muestra el interfaz de usuario que presenta. También se comenta la forma de ejecutar aplicaciones distribuidas en un sistema de fabricación añadiendo esta funcionalidad adicional al gestor.

C.1 Funcionalidad del Prototipo

El prototipo desarrollado se divide en dos partes bien diferenciadas, por un lado se ha desarrollado una aplicación gestora que permite realizar operaciones en el objeto gestionado, y por otro lado se ha desarrollado el objeto gestionado:

- Aplicación gestora: Presenta una interfaz de usuario que permite que un operador del sistema pueda llevar a cabo operaciones de gestión de una forma remota.
- Objeto gestionado: Presenta la interfaz correspondiente al servidor MMS que se ha comentado a lo largo de esta tesis, permitiendo la lectura y escritura de variables, la ejecución de programas, la notificación de eventos, etc. En el desarrollo

del objeto gestionado se ha tenido en cuenta la integración de este objeto en un sistema basado en CORBA, sin embargo no se ha tenido en cuenta la conexión con el Dispositivo de Fabricación Real. Todos los dispositivos de fabricación van a presentar la misma interfaz a la aplicación de gestión, sin embargo, internamente se debe llevar a cabo la traducción de los objetos del modelo de información MMS a las variables y objetos propios del dispositivo real.

El gestor se puede comunicar con un conjunto de objetos gestionados. Esta comunicación la realiza mediante el bus de objetos de CORBA. Aunque el prototipo utiliza un ORB básico y el mismo lenguaje de programación tanto en el gestor como en el objeto gestionado, la utilización de CORBA deja abierta la posibilidad de comunicación con objetos gestionados escritos en distintos lenguajes de programación y ejecutados en distintos sistemas operativos.

C.2 Plataforma de Implementación

Aquí se exponen las características software y hardware de la plataforma para la que se ha implementado el prototipo.

C.2.1 Lenguaje

El lenguaje elegido para el desarrollo ha sido Java, debido a las características con las que cuenta entre las que cabe mencionar las siguientes:

- Portabilidad: Debido a la concepción tan particular de Java, en donde se compila el código fuente y se obtiene un código especial llamado Bytecode que debe ser interpretado por la Máquina Virtual Java, los programas escritos en este lenguaje son portables, tanto a nivel de fuente como a nivel de bytecode, a cualquier máquina para la que se tenga disponible la máquina virtual.
- Orientado a Objetos: Java es un lenguaje orientado a objetos que proporciona los beneficios de esta tecnología como son la reutilización del código y la facilidad en la depuración, mantenimiento e integración.
- Sencillez: Java está basado en C++, que es un lenguaje familiar, pero elimina los elementos problemáticos de éste como pueden ser por ejemplo la utilización de punteros y su aritmética o la sobrecarga de operadores o las instrucciones de salto y proporciona una gestión de memoria automática eliminando los problemas de

la no liberación de memoria. Hay que considerar también que proporciona un conjunto de bibliotecas bastante potentes y fáciles de manejar como son por ejemplo el tratamiento de ventanas y la utilización de hilos.

Se ha utilizado el kit de desarrollo Java JDK 1.2 para la compilación. Se ha utilizado la librerías AWT (Advanced Windows Toolkit) para el desarrollo de la interfaz de usuario. Aunque en versiones anteriores estas librerías se encontraban en un Kit adicional al JDK, en la versión 1.2 ya son parte de éste.

C.2.2 Sistema Operativo

El prototipo se puede utilizar en cualquier sistema operativo que cuente con la Máquina Virtual Java. En concreto ha sido probado en Windows 95 y Solaris 2.5, tanto el cliente como el servidor. Pero hay que tener en cuenta que Java está disponible para una gran cantidad de sistemas operativos como por ejemplo Solaris 2.4, 2.5, 2.6, Windows 95, 98, NT 4.0, distintas distribuciones de Linux (Redhat Linux,Caldera Linux, Slackware Linux, S.u.S.E. Linux), OS/2 Warp 4.0, AIX y NetWare 5 entre otros.

C.2.3 ORB

La implementación del ORB utilizada ha sido JavaIDL [Sun99a], proporcionada por Sun de libre distribución y que sigue las especificaciones para CORBA 2.0. Debido a que esta implementación del ORB no cuenta con el servicio de eventos, como parte de este trabajo se ha abordado el desarrollo de este servicio con las funcionalidades básicas para llevar a cabo el prototipo.

Se podría haber utilizado cualquier ORB que proporcionara el mapeado de IDL a Java.

C.2.4 Plataforma Hardware

El ordenador utilizado ha sido un PC Pentium II a 266 MHz con 64M de memoria y una estación de trabajo Sun SparcStation 10.

C.3 Modelo de Objetos

El modelo de objetos ha sido especificado mediante la técnica de modelado de objetos OMT (Object Modeling Technique) de Rumbaugh [Rum95] . En el modelo de objetos se representan los aspectos estructurales de datos del sistema. En este modelo se pueden ver las clases de objetos que componen la aplicación así como las relaciones entre ellas.

La nomenclatura de símbolos utilizados en esta técnica son los que se muestran en la figura C.1.

C.3.1 Modelo de Objetos del Gestor

Se presenta en este apartado el modelo de objetos del gestor abreviado ya que no se incluyen todos los objetos, sólo aquellos que son más significativos. Por claridad sólo se ha incluido el nombre de los objetos, no los métodos ni los atributos de cada uno de ellos. Este diagrama se muestra en la figura C.2

Los objetos que se reflejan en el diagrama de objetos son los siguientes:

- Manager: Objeto que contiene la función principal del programa. Presenta la interfaz de usuario. Para ello se apoya en un conjunto de objetos que no están representados en el diagrama y que son los distintos paneles y botones que componen la interfaz. Se apoya en el objeto mmsCltMan para realizar todas las operaciones remotas sobre los objetos gestionados.
- mmsCltMan: Es el objeto que se encarga de gestionar la lista de clientes mms, también se encarga de crear el objeto mmsReportServer.
- mmsClient: Un objeto de este tipo representa un cliente mms. Este objeto es el que se encarga de realizar las peticiones correspondientes al objeto gestionado.
- mmsReportServer: Deriva de la clase Thread ya que este objeto debe ser un objeto activo. Se encarga de atender las peticiones que llegan desde el Dispositivo de Fabricación Virtual.
- mmsEventServer: Este objeto también deriva de la clase Thread. Se encarga de crear el objeto mmsEventServant.
- mms: Es la interfaz del objeto gestionado. Se incluyen en esta interfaz todos los métodos correspondientes al Dispositivo de Fabricación Virtual especificados en MMS.

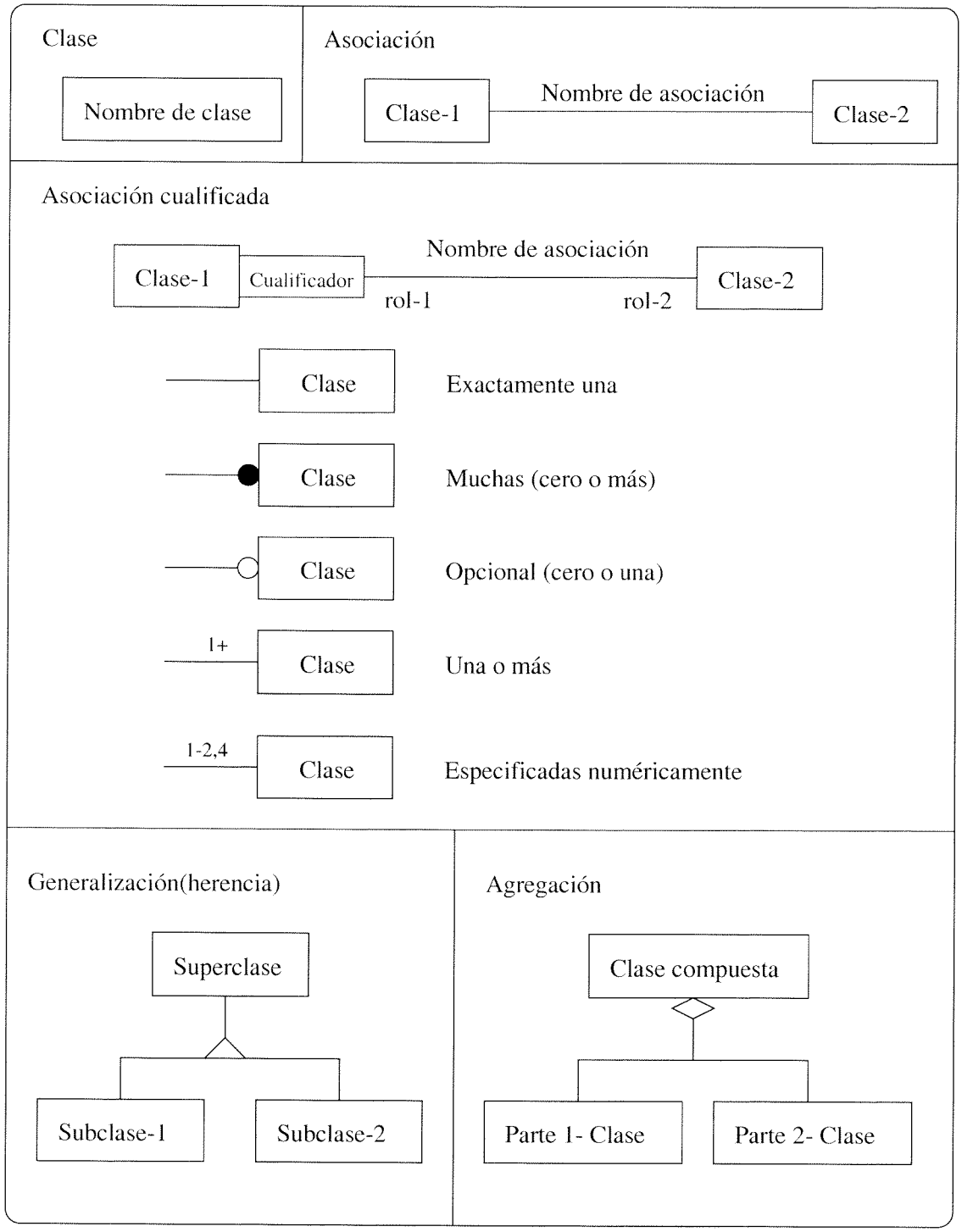


Figura C.1: Notación del Modelo de Objetos de OMT.

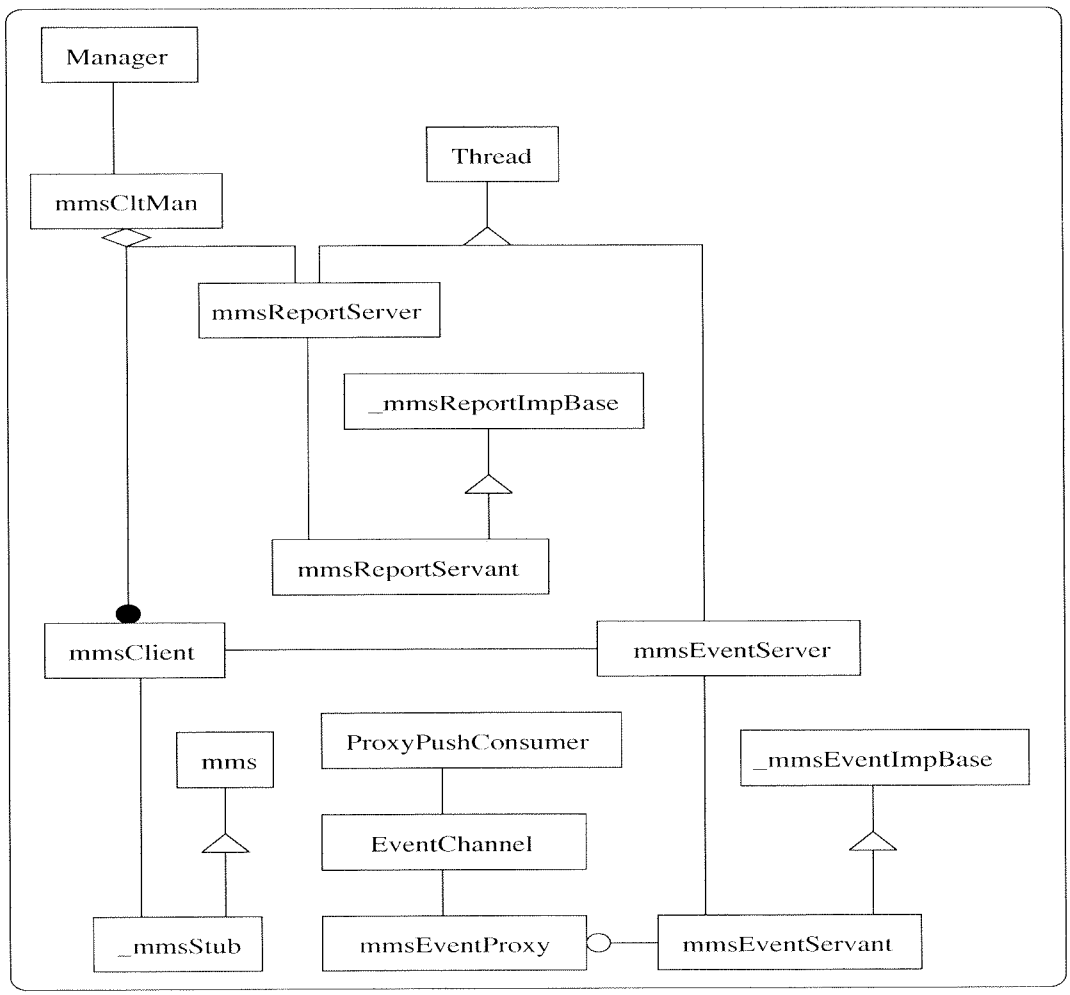


Figura C.2: Diagrama de Objetos del Gestor MMS.

- ProxyPushConsumer: Es el consumidor proxy para el modelo push. Es parte del canal de eventos de CORBA que se encarga de difundir los eventos.

- EventChannel: Es el canal de eventos definido en el modelo de eventos de CORBA.

- mmsEventProxy: Es el objeto que actúa como productor de eventos cuando llega una notificación de evento desde el objeto mmsServer.

- _mmsReportImpBase: Objeto correspondiente al esqueleto del servidor que muestra la interfaz mmsReport definida en IDL. La implementación de este objeto es automáticamente generada por el compilador de IDL idltojava comentado en el apartado correspondiente al mapeado de IDL a Java.

- _mmsEventImpBase: Objeto correspondiente al esqueleto del servidor que muestra la interfaz mmsEvent definida en IDL. La implementación de este objeto es

automáticamente generada por el compilador de IDL idltojava comentado en el apartado correspondiente al mapeado de IDL a Java.

- mmsReportServant: Este objeto deriva de la clase _mmsReportImpBase, e implementa los métodos de la interfaz mmsReport.

- mmsEventServant: Este objeto deriva de la clase _mmsEventImpBase, e implementa los métodos de la interfaz mmsEvent.

- _mmsStub: Implementa la interfaz mms. Es el soporte del cliente. La implementación de este objeto se obtiene automáticamente al compilar la especificación IDL del servidor.

- Thread: Es la clase de la biblioteca de Java que representa un hilo de ejecución.

C.3.2 Modelo de Objetos del Objeto Gestionado

Se presenta en este apartado el modelo de objetos abreviado del objeto gestionado. No se incluyen todos los objetos, sólo aquellos que son más significativos. Por claridad sólo se ha incluido el nombre de los objetos, no los métodos ni los atributos de cada uno de ellos. El diagrama se puede ver en la figura C.3

- mmsServer: Objeto que contiene la función principal del objeto servidor.

- mmsServant: Este objeto deriva de la clase _mmsImpBase, e implementa los métodos de la interfaz mms.

- _mmsImpBase: Objeto correspondiente al esqueleto del servidor que muestra la interfaz mms definida en IDL. La implementación de este objeto es automáticamente generada por el compilador de IDL idltojava comentado en el apartado correspondiente al mapeado de IDL a Java.

- mmsReport: Es la interfaz del objeto que atiende las peticiones del Dispositivo de Fabricación Virtual en el lado cliente. Se incluyen en esta interfaz todos los métodos MMS atendidos por el cliente, excepto la notificación de eventos.

- _mmsReportStub: Implementa la interfaz mmsReport. Es el soporte del cliente para el objeto mmsReport. La implementación de este objeto se obtiene automáticamente al compilar la especificación IDL del servidor.

- mmsEvent: Es la interfaz del objeto que atiende las notificaciones de eventos. En esta interfaz se incluye el método de notificación de eventos.

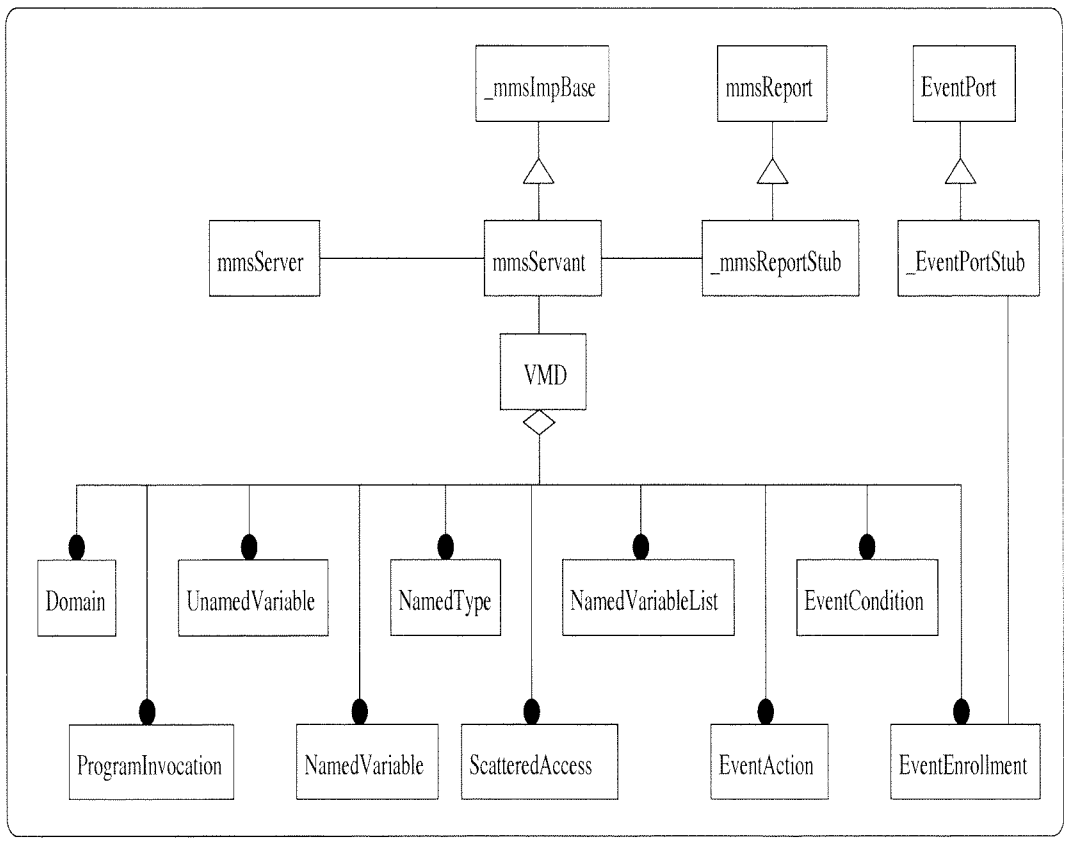


Figura C.3: Diagrama de Objetos del Servidor MMS.

- _mmsEventStub: Implementa la interfaz mmsEvent. Es el soporte del cliente para el objeto mmsEvent. La implementación de este objeto se obtiene automáticamente al compilar la especificación IDL del servidor.
- VMD: Objeto VMD especificado en MMS.
- Domain: Objeto Domain especificado en MMS.
- ProgramInvocation: Objeto ProgramInvocation especificado en MMS.
- UnamedVariable: Objeto UnamedVariable especificado en MMS.
- NamedVariable: Objeto NamedVariable especificado en MMS.
- NamedType: Objeto NamedType especificado en MMS.
- ScatteredAccess: Objeto ScatteredAccess especificado en MMS.
- NamedVariableList: Objeto NamedVariableList especificado en MMS.
- EventCondition: Objeto EventCondition especificado en MMS.
- EventAction: Objeto EventAction especificado en MMS.

- EventEnrollment: Objeto EventEnrollment especificado en MMS.

Los objetos especificados en MMS ya han sido introducidos en capítulos anteriores.

C.4 Procedimiento de Desarrollo

Para el desarrollo se han seguido una serie de fases hasta completar el prototipo. Las fases no son necesariamente secuenciales, ya que hay fases que se han ido completando a la vez. Las fases son las siguientes:

- Conversión de la especificación MMS en la interfaz IDL.
- Mapeado de la interfaz IDL a Java.
- Implementación del objeto gestionado.
- Implementación de la aplicación de gestión.

Conversión de MMS a IDL

El primer paso para el desarrollo de sistemas basados en CORBA es la identificación de las interfaces de los distintos objetos que intervienen en la aplicación. La interfaz IDL que se ha utilizado es una interfaz que se ajusta en lo posible al protocolo MMS. La adaptación de este protocolo a IDL se ha expuesto con detalle en el capítulo 6: "Adaptación de MMS a CORBA", donde se introduce la especificación de traducción de ASN.1 a IDL que se ha utilizado, así como los distintos ajustes que se han realizado a la hora de la adaptación.

Mapeado de la Interfaz IDL a Java

Una vez que se dispone de la interfaz IDL de los objetos CORBA que se van a utilizar en la aplicación, se realiza el mapeado de esta interfaz al lenguaje en el que se va a desarrollar la aplicación, en este caso Java. La traducción de la interfaz especificada en IDL a las construcciones utilizadas en el lenguaje Java sigue la especificación del mapeado de IDL al lenguaje Java proporcionada por la OMG en el capítulo correspondiente a este mapeado de la especificación CORBA [Obj95b], aunque en la última

versión se puede encontrar como un documento separado [Obj99]. De este mapeado cabe destacar los siguientes aspectos:

- Nombres: En general los nombres e identificadores IDL son traducidos a nombres e identificadores Java sin cambio. Si existe algún conflicto con alguna palabra reservada o con algún otro nombre de clase entonces se antepone un carácter de subrayado para evitar el conflicto.

- Mapeado de módulos: Un módulo IDL es mapeado a un paquete Java con el mismo nombre. Todas las declaraciones de tipos en el módulo son mapeadas a las correspondientes declaraciones de clases o interfaces Java en el paquete generado.

- Mapeado de los tipos básicos: La tabla C.1 muestra el mapeado de los tipos básicos de IDL a Java. En los casos en los que se puede producir un error al realizar la conversión (por ejemplo cuando el rango del tipo Java es mayor que el de IDL), se muestra la excepción correspondiente.

IDL Type	Java Type	Exceptions
boolean	boolean	
char	char	CORBA::DATA_CONVERSION
wchar	char	CORBA::DATA_CONVERSION
octet	byte	
string	java.lang.String	CORBA::MARSHAL CORBA::DATA_CONVERSION
wstring	java.lang.String	CORBA::MARSHAL CORBA::DATA_CONVERSION
short	short	
unsigned short	short	
long	int	
unsigned long	int	
long long	long	
unsigned long long	long	
float	float	
double	double	

Tabla C.1: Mapeado de tipos básicos

- Mapeado de otros tipos: En la especificación también se proporciona el mapeado para el tipo enumerado (enum), estructura (struct), unión (union), secuencia (sequence) y vectores (array). Las construcciones que se crean durante el mapeado a Java de cada uno de estos tipos se pueden resumir de la siguiente forma:

  - enum: Se crea una clase Java con el mismo nombre que el enumerado. La clase contiene un método llamado value que devuelve el valor entero de la instancia del enumerado. Este valor es asignado secuencialmente comenzando por 0. Contiene además miembros que permiten la utilización de las etiquetas del enumerado.
  - struct: Se crea una clase Java con el mismo nombre que la estructura. La clase contiene miembros públicos correspondientes a cada uno de los campos de la estructura.
  - union: Se crea una clase Java con el mismo nombre que la unión. La clase contiene miembros para acceder al discriminador, y para acceder o modificar cada una de las ramas de la unión incluyendo la opción por defecto.
  - sequence: Se mapea a un array con el mismo nombre.
  - array: Se mapea a un array con el mismo nombre.

- Mapeado de interfaces: Una interfaz IDL se mapea a una interfaz Java con las mismas operaciones.

- Mapeado de excepciones: Las excepciones son mapeadas de una forma similar a las estructuras e implementan la interfaz org.omg.CORBA.UserException.

- Mapeado de Typedef: En Java no existe la definición de tipos, por eso, las declaraciones de tipos simples son mapeadas al tipo original, y para las declaraciones de tipos complejos, se traducen hasta conseguir tipos simples o tipos definidos en IDL. Para las secuencias y los arrays se generan clases Holder y son éstas las que se utilizan en la traducción.

- Paso de parámetros: Los parámetros de entrada son mapeados a parámetros normales Java. Los resultados de la operación IDL son devueltos como resultados del correspondiente método Java. Para la utilización de parámetros de salida o entrada/salida (out o inout) se necesitan las clases Holder.

- Clase Helper: Todos los tipos IDL definidos por el usuario tienen una clase Java adicional con el sufijo Helper añadido al nombre de tipo generado. Se proporcionan en esta clase métodos necesarios para manipular el tipo definido. Entre estos métodos se incluyen las operaciones para insertar y extraer un dato de tipo Any del tipo correspondiente, obtener un identificador para el repositorio, obtener un código de tipo, leer y escribir el tipo desde y a un flujo y la operación narrow para la conversión de un objeto a este tipo.

- Clase Holder: Se requiere para soportar el paso de parámetros de salida (out) o de entrada/salida (inout) ya que Java sólo contempla el paso de parámetros por valor. Estas clases están disponibles para todos los tipos básicos IDL y deben ser generadas para todos los tipos definidos por el usuario. El nombre de la clase holder es construido añadiendo Holder al nombre del tipo. Cada clase holder

contiene un constructor por defecto, un constructor con un parámetro del tipo correspondiente, y un miembro público llamado value que contiene el valor tipado.

El mapeado se realiza de forma automática utilizando la herramienta idltojava proporcionada por JavaIDL. Esta herramienta toma como entrada el fichero con la especificación IDL y genera un conjunto de clases. Éstas son las siguientes:

- Stub: Clase abstracta que proporciona la funcionalidad CORBA del cliente.
- Skeleton: Clase abstracta que proporciona la funcionalidad CORBA del servidor.
- Clases Holder: Clases que permiten el paso de parámetros de salida y entrada/salida de los distintos tipos de datos especificados en IDL.
- Clases Helper: Clases que contienen operaciones auxiliares para los distintos tipos de datos especificados en IDL.
- Interfaz de los objetos servidores: Interfaz que contiene la versión Java de la interfaz IDL especificada.

A partir de aquí ya se puede comenzar con la implementación del objeto gestionado y la aplicación de gestión ya que se dispone de todos los elementos clave para la comunicación entre los distintos objetos que se encuentran en el bus de objetos.

Implementación del Objeto Gestionado

Una vez que se ha realizado el mapeado de IDL a Java y se cuenta con todas las clases necesarias para el desarrollo de la aplicación, se debe realizar la implementación de las operaciones de las interfaces definidas en la especificación IDL. Para ello se crean las clases que implementan cada una de las interfaces generadas a partir del mapeado. Hay que proporcionar entonces el cuerpo de cada uno de los métodos que aparece en la interfaz.

El modelo de objetos de esta parte de la aplicación ha sido dado en el apartado "Modelo de Objetos del Objeto Gestionado". Se debe implementar la interfaz mms que contiene los métodos correspondientes al Dispositivo de Fabricación Virtual.

Implementación de la Aplicación de Gestión

En este punto, ya se puede crear la aplicación que haga uso de los objetos CORBA que se especifican en IDL. El uso de estos objetos se realiza de la misma forma que si de objetos Java normales se tratara, es decir, una vez realizada la declaración y creación del objeto se puede pasar a la ejecución de sus métodos, que desde el punto de vista de la aplicación se hará de una forma transparente, sin tener en cuenta que los métodos son remotos y no locales.

El modelo de objetos de esta parte de la aplicación ha sido dado en el apartado "Modelo de Objetos del Gestor". Se debe implementar la interfaz mmsReport y la interfaz mmsEvent.

C.5 Servidor de Nombres

El servidor de nombres es el encargado de almacenar las referencias a los objetos que se encuentran en el bus de objetos, haciendo la correspondencia entre el nombre del objeto y su referencia. En JavaIDL el servidor de nombres es tnameserv.

La aplicación de gestión utiliza el servidor de nombres para localizar los objetos gestionados requeridos por el usuario. El usuario debe proporcionar el nombre y la aplicación obtiene la referencia haciendo uso de este servicio. Una vez obtenida la referencia, se pueden llamar a los métodos del objeto.

En la parte del objeto gestionado también es necesario utilizar este servicio cuando la referencia a la aplicación a la que hay que notificar los eventos contiene el nombre del objeto. También hay que utilizar este servicio cuando se conozca el nombre y no la referencia del objeto al que hay que enviar las peticiones no solicitadas. La forma de obtener este nombre o referencia es una cuestión interna al VMD.

C.6 Interfaz de Usuario

La interfaz de usuario mostrada por la aplicación de gestión es una interfaz sencilla que permite enlazar con los objetos gestionados, y una vez enlazados permite la ejecución de operaciones MMS en el dispositivo de fabricación virtual.

La figura C.4 muestra la pantalla principal de la interfaz de usuario, donde se puede

indicar el nombre del dispositivo al que se quiere enlazar, una vez enlazado, se muestra de forma permanente (hasta que no sea eliminado explícitamente por el usuario) un icono con el nombre de este dispositivo, seleccionando este icono se pueden realizar cuantas operaciones se crean oportunas.

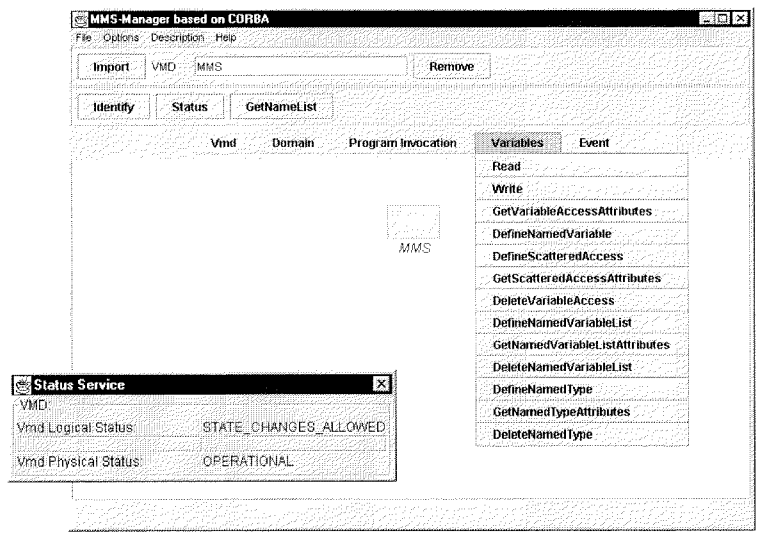


Figura C.4: Interfaz del Gestor MMS.

C.7 Ejecución de Aplicaciones Distribuidas

Una vez desarrollado el prototipo del gestor y el objeto gestionado, se cuenta con todo lo necesario para añadir funcionalidades adicionales al gestor.

Una de estas funcionalidades es la ejecución de aplicaciones distribuidas [Ari00] en el sistema de fabricación, iniciadas y supervisadas por el gestor.

Una aplicación distribuida está compuesta por diversas tareas llevadas a cabo en diferentes componentes del sistema distribuido. Debe permitirse la ejecución en paralelo de distintas tareas al mismo tiempo. También se deben contemplar puntos de sincronización entre los distintos componentes.

Los distintos componentes del sistema se van a modelar como Dispositivos de Fa-

bricación Virtual (VMD). La comunicación con estos componentes se va a llevar a cabo a través del ORB de CORBA por medio de peticiones MMS. La activación de tareas se realiza usando los servicios asociados al objeto Invocación de Programa contemplado en este estándar. Las tareas son comenzadas como resultado de las notificaciones de eventos recibidas de los distintos componentes.

Desde el punto de vista de la ejecución de aplicaciones distribuidas, éstas se ven como un conjunto de tareas. Cada tarea es llevada a cabo por un VMD concreto. Cada VMD puede generar notificaciones de eventos que pueden a su vez activar nuevas tareas en otros componentes.

Para especificar el comportamiento del sistema se han usado las Redes de Petri, ya que son una herramienta de especificación que permiten modelar acciones concurrentes y sincronización. Se han usado las Redes de Petri Binarias debido a su simplicidad y fácil implementación. A cada aplicación distribuida se le asocia una Red de Petri que refleja el comportamiento del sistema.

El comienzo de la ejecución de una tarea se ha modelado como un lugar en la Red de Petri. Por otra parte, una notificación de evento es modelada como una transición. Cuando la notificación de evento se produce, la transición podrá ser disparada en el caso de que los lugares de entrada estén marcados.

Se ha introducido un componente supervisor capaz de llevar a cabo la ejecución de la aplicación distribuida teniendo en cuenta la Red de Petri asociada a esta aplicación.

El Objeto Supervisor

El objeto supervisor es el responsable de la activación de las distintas tareas de la aplicación distribuida teniendo en cuenta el comportamiento que debe tener el sistema reflejado en la Red de Petri.

Las responsabilidades de este objeto son las siguientes:

- Construir los objetos Invocación de Programa en los distintos VMDs. Como regla general estos objetos se crearán indicando que son invocaciones de programas monitorizadas. Es una posibilidad que permite MMS y que significa que se enviará una notificación de evento a la aplicación especificada en el objeto invocación de programa cuando la ejecución del programa termina.

- Comenzar las diferentes tareas en los distintos VMDs cuando sea necesario.

- Recibir las notificaciones de eventos enviadas por los VMDs.
- Controlar el estado del sistema, actualizando este estado conforme a las notificaciones recibidas y ordenando la ejecución de tareas en concordancia con la evolución del estado del sistema.

Toda la comunicación entre este objeto supervisor y los VMDs se lleva a cabo utilizando el bus de objetos de CORBA. El supervisor actúa como un cliente MMS que puede requerir los servicios del VMD.

El componente supervisor debe actualizar el estado de la Red de Petri. Esto quiere decir, que debe actualizar el estado de una transición cuando llegue una notificación de evento correspondiente por parte de algún VMD. Por otra parte debe comprobar si la Red de Petri puede evolucionar después de este cambio en la transición. Si la transición es disparable y la notificación ha llegado, el disparo se lleva a cabo. Este disparo trae como resultado el marcado de los lugares de salida de esta transición que equivale al comienzo de las tareas asociadas a estos lugares.

En resumen, el objeto supervisor debe controlar la evolución de la Red de Petri, disparando las transiciones cuando ocurren los eventos asociados y comenzando las tareas asociadas a los lugares.

Una Aplicación

En esta sección se pretende dar una visión general de la funcionalidad del supervisor mediante una aplicación distribuida de ejemplo.

El sistema de fabricación donde se va a ejecutar la aplicación se muestra en la figura C.5. El sistema distribuido de fabricación está compuesto por distintos dispositivos (robots, cinta, almacén, máquina de control numérico, etc). La aplicación distribuida va a ser ejecutada en este sistema y estará compuesta por distintas tareas llevadas a cabo por los distintos dispositivos.

La aplicación consiste de un conjunto de tareas que permitirán sacar piezas del almacén de dos tipos diferentes (tipo 1 y tipo 2), procesar estas piezas, ensamblarlas y devolverlas al almacén una vez ensambladas.

El robot que controla el almacén tendrá que poner las piezas en los pallets, alternando piezas tipo 1 y tipo 2. Una vez que una pieza está en el pallet, el autómata que es responsable de mover la cinta tiene que transportar la pieza al lugar adecuado.

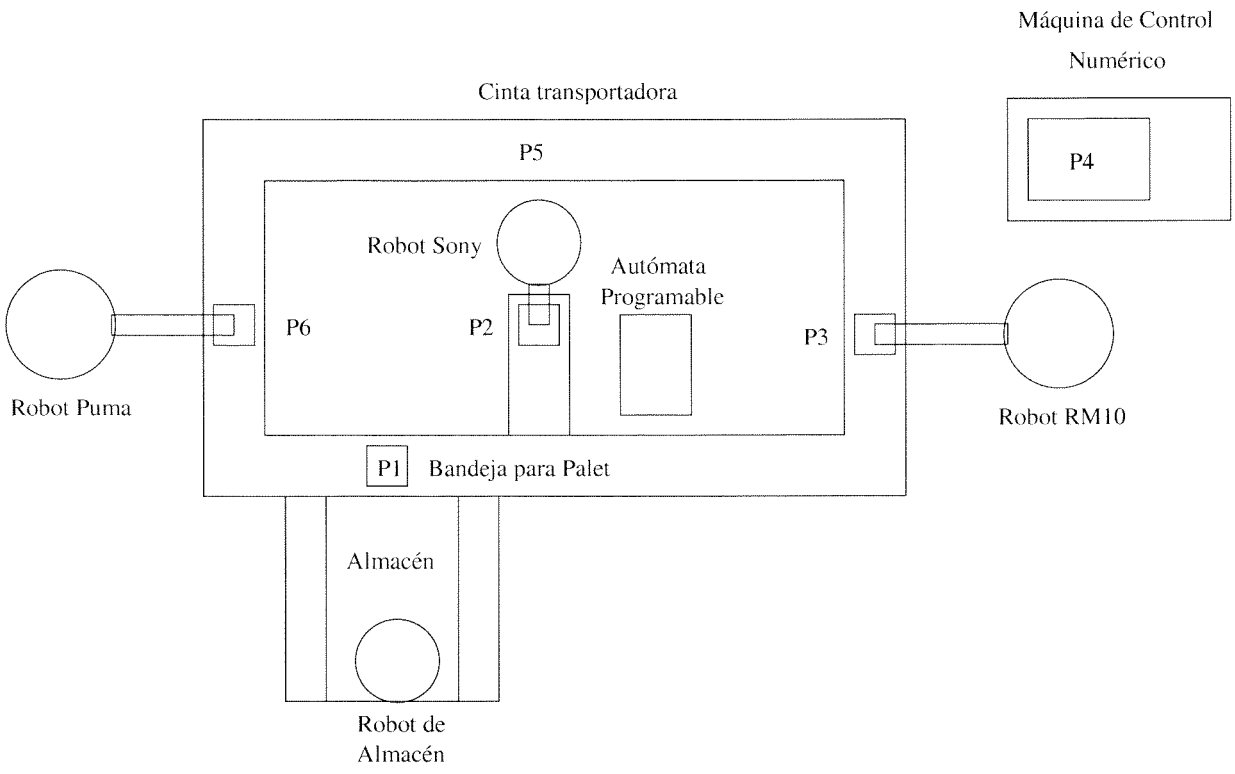


Figura C.5: Sistema de Fabricación.

El robot de almacén tiene además que recoger las piezas ensambladas que llegan y guardarlas en el almacén.

La pieza tipo 1 tiene que ser procesada por la máquina de control numérico. Por lo tanto, es llevada a la posición P3, donde el robot RM10 tiene que ponerla dentro de la máquina de control numérico donde es procesada, y llevarla de vuelta a la cinta para ser transportada hasta la posición P5.

Por otro lado, la pieza tipo 2 es llevada a la posición P2. Es procesada por el SONY. Cuando esta pieza ha sido procesada y la pieza tipo 1 ha llegado a P5 se realiza el ensamblado.

A continuación, el PUMA debe chequear si la nueva pieza es correcta o no. Si la pieza es correcta, se lleva de vuelta al almacén. En otro caso, el robot debe tirar esta pieza a un contenedor de reciclado.

La Red de Petri que muestra el comportamiento se muestra en la figura C.6. Como se refleja en la red, hay tareas que se pueden ejecutar de forma concurrente teniendo en cuenta ciertos puntos de sincronización. La aplicación se ha simplificado en lo posible para que sea más clara. No se han considerado las situaciones de funcionamiento inco-

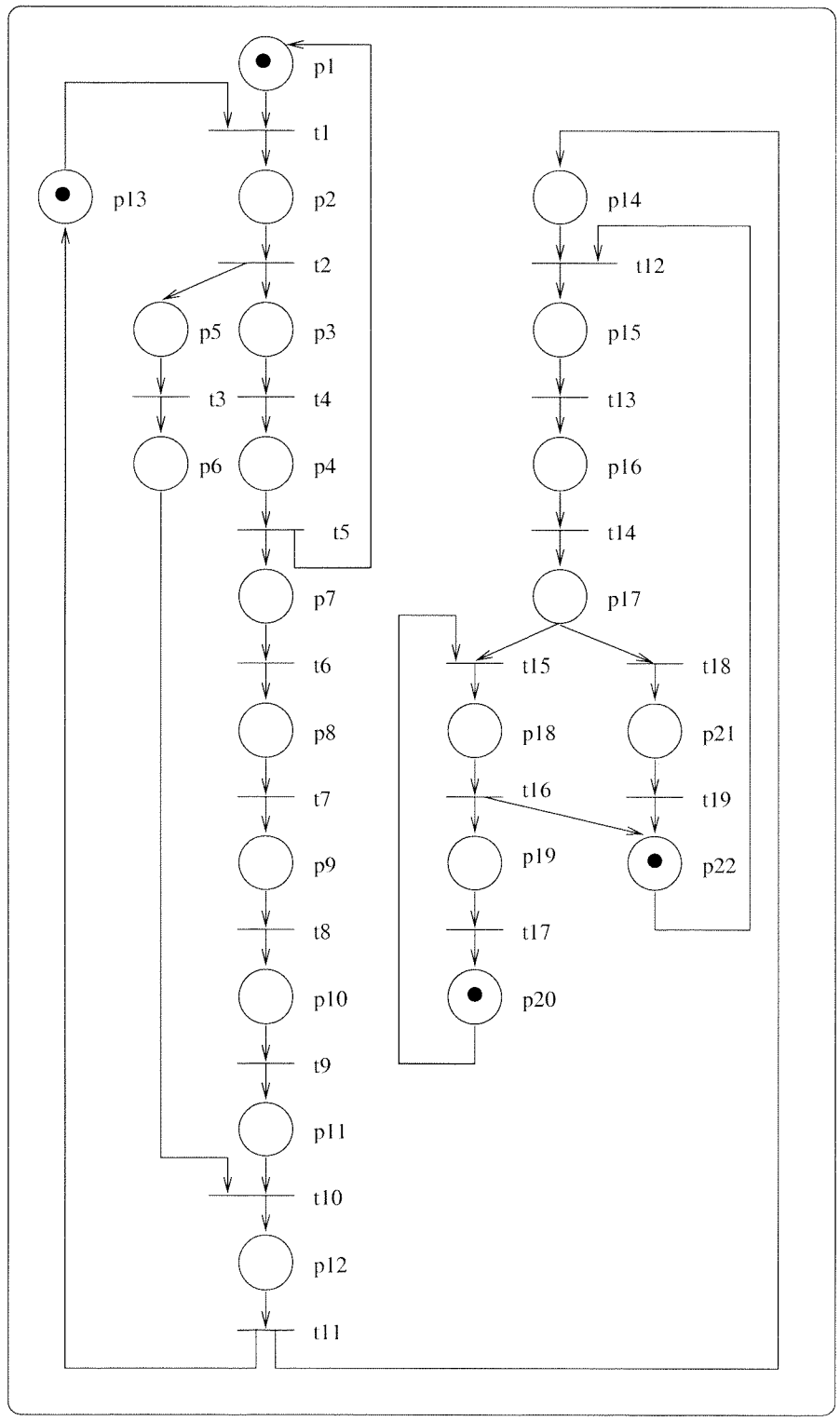


Figura C.6: Red de Petri.

rrecto, pero en todo caso, la red puede ser ampliada para que considere la situaciones que se crean convenientes.

Las diferentes tareas que componen la aplicación son las siguientes:

- P1WAREHOUSE: Mueve la pieza tipo 1 desde el almacén al pallet.
- P2WAREHOUSE: Mueve la pieza tipo 2 desde el almacén al pallet.
- P3WAREHOUSE: Mueve la pieza desde el pallet al almacén.
- PijCONVEYOR: Mueve el pallet desde la posición Pi a la posición Pj.
- P1SONY: Procesa la pieza tipo 2.
- P2SONY: Ensambla la pieza tipo 1 y tipo 2.
- P1RM10: Mueve la pieza tipo 2 desde la posición P3 a la posición P4.
- P2RM10: Mueve la pieza tipo 2 desde la posición P4 a la posición P3.
- P1NCM: Procesa la pieza tipo 2.
- P1PUMA: Chequea la nueva pieza.
- P2PUMA: Tira la pieza.

La ejecución de las tareas en los distintos componentes y la espera hasta llevar a cabo la sincronización han sido asociadas a los lugares. Éstos se muestran en la figura C.7.

p1: Start P1WAREHOUSE	p12: Start P2SONY
p2: Start P12CONVEYOR	p13: Wait
p3: Start P2WAREHOUSE	p14: Wait
p4: Start P13CONVEYOR	p15: Start P56CONVEYOR
p5: Start P1SONY	p16: Start P1PUMA
p6: Wait	p17: Wait
p7: Start P1RM10	p18: Start P61CONVEYOR
p8: Start PNCM	p19: Start P3WAREHOUSE
p9: Start P2RM10	p20: Wait
p10: Start P35CONVEYOR	p21: Start P2PUMA
p11: Wait	p22: Wait

Figura C.7: Lugares.

Las notificaciones de eventos ocurren cuando las tareas finalizan y cuando el PUMA detecta si la pieza es correcta o no.

Las transiciones corresponden a la comunicación de eventos llevada a cabo por los dispositivos de fabricación. Se han introducido nuevas transiciones cuando se necesita realizar una sincronización. Las transiciones de la red se muestran en la figura C.8.

t1: Notification: P1WAREHOUSE ended
t2: Notification: P12CONVEYOR ended
t3: Notification: P1SONY ended
t4: Notification: P2WAREHOUSE ended
t5: Notification: P13CONVEYOR ended
t6: Notification: P1RM10 ended
t7: Notification: PNCM ended
t8: Notification: P2RM10 ended
t9: Notification: P35CONVEYOR ended
t10: Synchronization
t11: Notification: P2SONY ended
t12: Synchronization
t13: Notification: P56CONVEYOR ended
t14: Notification: P1PUMA ended
t15: Notification: CORRECTPIECE
t16: Notification: P61CONVEYOR ended
t17: Notification: P3WAREHOUSE ended
t18: Notification: NOCORRECTPIECE
t19: Notification: P2PUMA

Figura C.8: Transiciones.

El objeto supervisor toma como entrada la Red de Petri. La red se especifica como un conjunto de transiciones, para cada transición, un conjunto de lugares de entrada y un conjunto de lugares de salida, y una serie de marcas iniciales. También se le da como entrada al supervisor el conjunto de programas que son ejecutados en cada dispositivo, para poder crear los objetos Invocación de Programa en cada uno de ellos.

Para llevar a cabo la ejecución de la aplicación distribuida, el primer paso que se da es iniciar las tareas asociadas a los lugares marcados en el marcado inicial. A la llegada de las notificaciones de eventos, el objeto supervisor actualiza el estado de la Red de Petri e inicia las tareas asociadas a los lugares que han sido marcados en el nuevo estado.

De esta forma, la aplicación distribuida se ejecuta, haciendo uso de los servicios

MMS correspondientes a las Invocaciones de Programas y de Eventos, según establezca la Red de Petri correspondiente a la evolución del sistema.

Bibliografía

[Ari98a] Ariza, T., Rubio, F.R. *Communicating MMS Events in a Distributed Manufacturing System using CORBA.* Preprints DCCS'98, 1998.

[Ari98b] Ariza, T., Rubio, F.R. *MMS-Manager: Device Management in Heterogeneous Environment Based on CORBA.* Preprints Controlo'98, 1998.

[Ari99a] Ariza, T., Madinabeitia, G., Ternero, J.A. *CAPISERVER: Remote Access to ISDN Using CORBA.* Preprints AI'99, 1999.

[Ari99b] Ariza, T., Rubio, F.R. *Notifying MMS Events to CORBA Objects.* Preprints MIC'99, 1999.

[Ari00] Ariza, T., Rubio, F.R. *Running Distributed Applications Using MMS-CORBA.* WFCS 2000, 2000.

[Ase98] Asensio, J.I., Villagrá, V.A., Berrocal, J. *A Simplified Approach to the Management of a CORBA based Electronic Brokerage Application.* OVUA'98, 1998.

[BMC99] BMC. *Making WBEM Work for You.* DMTF Annual Conference, 1999.

[Cek98] Cekro, Z. *Simple Network Management Protocol (SNMP) -Current Standards and Status-.* Service Télématique et Communication, 1998.

[Chu97] Chung, P.E. *DCOM and CORBA Side by Side, Step by Step, and Layer by Layer.* http://www.bell-labs.com/~ emerald /dcom_corba/Paper.html, 1997.

[Cor91] Corbin, John R. *The Art of Distributed Applications.* Springer-Verlag, 1991.

[Dis99] Distributed Management Task Force. *CIM Specification v2.2.* DMTF, 1999.

[ESP95] ESPRIT Consortium CCE-CNMA. *MMS: A Communication Language for Manufacturing.* Springer, 1995.

[Fin94] Finin, R., Fritzson, R., McKay, D. and McEntire, R. *KQML as an agent Communication Language.* Proc. Conf. on Information and Knowledge Management (CIKM'94, 1994.

[Gen97] Genilloud, G. *Integrating the OSI and the CORBA Architectures for Systems Management.* Proceedings of the CORBA Management Workshop at the OMG TC Meeting in Dublin(Ireland), 1997.

[Gen98] Genilloud, G. *Integrating the OSI and the Distributed Objects Architectures for Systems Management.* IEEE/IFIP Network Operations and Management Symposium, 1998.

[Góm00] Gómez Bocanegra, José Luis. *Diseño e Implementación de un Gestor SNMP Basado en CORBA.* PFC desarrollado en el A.I.Telemática de la E.S.I de Sevilla dirigido por T.Ariza, 2000.

[Gol95] Goldszmidt, G. and Yemini, Y. *Distributed Management by Delegation.* Proc. Int. Conf. on Distributed Computing Systems (ICDCS'95), 1995.

[Gre95] Gressier-Soudan, E., Lefebvre, M., Natkin, S. *TCP/IP Manufacturing Applications: an experiment with MMS over RPC.* ULPAA Web Conference, 1995.

[Guy97] Guyonnet, G., Gressier-Soudan, E., Weis, F. *COOL-MMS: a CORBA approach for ISO-MMS.* ECOOP'97 WorkShop, 1997.

[Har97] Harrison, T.H., Levine, D.L. and Schmidt, D.C. *The Design and Performance of a Real-Time CORBA Object Event Service.* OOPSLA'97, 1997.

[Inc91] Ince, Darrel. *Object-Oriented Software Engineering With C++.* McGraw-Hill, 1991.

[ISO90a] ISO/IEC. 9506-1. *Industrial Automation Systems - Manufacturing Message Specification", Part 1: Service Definition.* International Standards Organization, 1990.

[ISO90b] ISO/IEC. 9506-2. *Industrial Automation Systems - Manufacturing Message Specification", Part 2: Protocol Specification.* International Standards Organization, 1990.

[ISO94a] ISO/IEC 10746-1. *Information Technology - Basic reference model of Open Distributed Processing - Part 1: Overview and guide to use.* International standard, 1994.

[ISO94b] ISO/IEC 10746-2. *Information Technology - Basic reference model of Open Distributed Processing - Part 2: Descriptive Model.* International standard, 1994.

[ISO94c] ISO/IEC 10746-3. *Information Technology - Basic reference model of Open Distributed Processing - Part 3: Prescriptive Model.* International standard, 1994.

[ISO94d] ISO/IEC 10746-4. *Information Technology - Basic reference model of Open Distributed Processing - Part 4: Architectural Semantics.* International standard, 1994.

[ITU92a] ITU-T. *Information technology Open Systems Interconnection Procedures for the operation of OSI Registration Authorities General procedures.* Recommendations X.660 ITU-T, 1992.

[ITU92b] ITU-T. *Information technology Open Systems Interconnection Structure of Management Information: Guidelines for the definition of managed objects.* Recommendations X.722 ITU-T, 1992.

[ITU96] ITU-T. *Principles for Telecommunications Management Networks.* Recommendations M.3010 ITU-T, 1996.

[ITU97a] ITU-T. *Information technology Open Systems Interconnection Common Management Information Service.* Recommendations X.710 ITU-T, 1997.

[ITU97b] ITU-T. *Information technology Open Systems Interconnection Common Management Information Protocol: Specification.* Recommendations X.711 ITU-T, 1997.

[ITU97c] ITU-T. *Information technology Open Systems Interconnection Systems management overview.* Recommendations X.701 ITU-T, 1997.

[ITU97d] ITU-T. *Information technology Open Systems Interconnection Systems management: Command sequencer for systems management.* Recommendations X.753 ITU-T, 1997.

[Kel98] Keller, A. *Towards Interoperable Management Architectures: Making SNMP Management Platforms suitable for CORBA.* OVUA'98, 1998.

[Lau98] Laurent, André. *OO-MMS: Approche CORBA/Java pour ISO-MMS.* Présentation André Laurent/CNAM-CEDRIC, 1998.

[Lev97] Levi, D.B. and Schonwalder, J. *Script MIB. Definitions of Managed Objects for the Delegation of Management Scripts. Internet draft.* IETF, 1997.

[Mar98] Martin-Flatin, JP., Znaty, S. , Hubaux, JP. *A Survey of Distributed Network and Systems Management Paradigms.* Techical Report SSC/1998/024, 1998.

[Obj95a] Object Management Group. *Common Facilities Architecture.* OMG, Framingham, MA., 1995.

[Obj95b] Object Management Group. *CORBA: Common Object Request Broker Architecture and Specification.* OMG, Framingham, MA., 1995.

[Obj95c] Object Management Group. *CORBAservices: Common Object Services Specification.* OMG, Framingham, MA., 1995.

[Obj99] Object Management Group. *IDL to Java Language Mapping Specification.* OMG, 1999.

[OMG98] OMG. *JIDM Interaction Translation.* OMG Document Number: telecom/98-10-10, 1998.

[Ope97] Open Group. *Inter-domain Management: Specification Translation.* X/Open Document Number: P509, 1997.

[Orf96] Orfali, R., Harkey, D., Edwards, J. *The Essential Distributed Objects. Survival Guide.* Wiley, 1996.

[Pim90] Pimentel, Juan R. *Communication Networks for Manufacturing.* Prentice-Hall, 1990.

[Ram99] Ramírez Retamal, Mª Elena. *Diseño e Implementación de un Agente Proxy entre SNMP y CORBA.* PFC desarrollado en el A.I.Telemática de la E.S.I de Sevilla dirigido por T.Ariza, 1999.

[Rec94a] Recomendación UIT-T X.680. *Notación de Sintaxis Abstracta Uno: Especificación de la Notación Básica.* UIT-T, 1994.

[Rec94b] Recomendación UIT-T X.690. *Especificación de las Reglas de Codificación Básica, de las Reglas de Codificación Canónica y de las Reglas de Codificación Distinguida.* UIT-T, 1994.

[RFC90a] RFC1155. *Structure and Identification of Management Information for TCP/IP-based internets.* 1990.

[RFC90b] RFC1157. *Simple Network Management Protocol (SNMP)* . 1990.

[RFC91a] RFC1212. *Concise MIB Definitions.* 1991.

[RFC91b] RFC1213. *Management Information Base for Network Management of TCP/IP-based internets: MIB II.* 1991.

[RFC97] RFC2126. *ISO Transport Service on top of TCP.* 1997.

[Rum95] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. *Modelado y Diseño Orientados a Objetos. Metodología OMT.* Prentice Hall, 1995.

[Sol95] Soley, R.M., Stone, C.M. *Object Management Architecture Guide.* John Wiley and Sons, 1995.

[Sun97] Sun Microsystems, Inc. *RMI and Java Distributed Computing.* http://www.javasoft.com/features/1997/nov/rmi.html, 1997.

[Sun99a] Sun Microsystems, Inc. *Java IDL.* http://www.javasoft.com/products/jdk/1.2/docs/guide/idl, 1999.

[Sun99b] Sun Microsystems, Inc. *Java Management Extensions.* Sun Microsystems, Inc., 1999.

[Sun99c] Sun Microsystems, Inc. *Java Management Extensions. CIM/WBEM Manager APIs.* Sun Microsystems, Inc., 1999.

[Sun99d] Sun Microsystems, Inc. *Java Management Extensions. SNMP Manager API.* Sun Microsystems, Inc., 1999.

[Sun99e] Sun Microsystems, Inc. *Java Management Extensions White Paper.* Sun Microsystems, Inc., 1999.

[Tan95] Tanenbaum, Andrew S. *Distributed Operating Systems.* Prentice-Hall, 1995.

[The96] The Open Group. *DCE Distributed Computing Environment.* http://www.opengroup.org/dce/info/papers/tog-dce-pd-1296.htm, 1996.

[TIN97] TINA-C. *Overall Concepts and Principles of TINA.* TINA-C, 1997.

[UT97] UIT-T. *Tecnología de la Información - Arquitectura de Gestión Distribuida Abierta.* Recomendación UIT-T X.703, 1997.

[UT98] UIT-T. *Tecnología de la Información - Arquitectura de Gestión Distribuida Abierta. Enmienda 1: Soporte de Arquitectura de Negociación de petición de Objetos Comunes.* Recomendación UIT-T X.703 Enmienda 1, 1998.

[Val92] Valenzano, A., Demartini, C., Ciminiera, L. *MAP and TOP Communications.* Addison Wesley, 1992.

[Val99] Valera, F., Moreno, J.I., Villagrá, V.A., Berrocal, J. *Mecanismos de Comunicación y Gestión de Servicio de un Broker de Información Multiagentte.* JITEL'99, 1999.

[Vin98] Steve Vinoski. *New Features For CORBA 3.0.* Communications of the ACM, October, 1998.

[Wil99] Williams, Ray. *WBEM.* DMTF Annual Conference, 1999.

[Woo95] Wooldridge, M. and Jennings, N.R. *Agent Theories, Architectures and Languages: a Survey.* Workshop on Agent Theories, Architectures and Languages, 1995.

[Zuk98] Zukowski, John. *Introduction to the JavaBeans API.* MageLang Institute. http://developer.java.sun.com/developer/onlineTraining/Beans/JBeansAPI/index.html, 1998.