UNIVERSIDAD AUSTRAL DE CHILE
FACULTAD DE CIENCIAS DE LA INGENIERÍA
ESCUELA DE INGENIERÍA CIVIL EN INFORMÁTICA

DESARROLLO DE UNA METODOLOGÍA DE INTEGRACIÓN DE SISTEMAS BASADO EN EL MONITOR TRANSACCIONAL BEA TUXEDO

TESIS DE GRADO PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Patrocinante:
Ing. Gladys Mansilla.

Copatrocinante:
Ing. Patricio Vera.

Karina A. Herrera H.
Claudio J. Altamirano A.
VALDIVIA – CHILE
2003

AGRADECIMIENTOS

A DIOS por su sabiduría.
A mi madre por su amor y sabios consejos.
A mis amores Marcia y Valentina por su paciencia y comprensión.
A mis hermanos por su constante apoyo.
A Karina por su compañerismo y esfuerzo.
Claudio J. Altamirano A.

A Dios y la Virgen por darme la posibilidad de cumplir una meta inconclusa.
A mis papás por tanta paciencia, sacrificios y amor que me han entregado.
A mis hermanos por creer en mí.
A Claudio por tener siempre fe que lo lograríamos.
Karina A. Herrera H.

VALDIVIA, 11 DE JULIO DEL 2003

DE: GLADYS MANSILLA GOMEZ
A : DIRECTORA DE ESCUELA INGENIERIA CIVIL EN INFORMATICA

MOTIVO

INFORME TRABAJO DE TITULACION

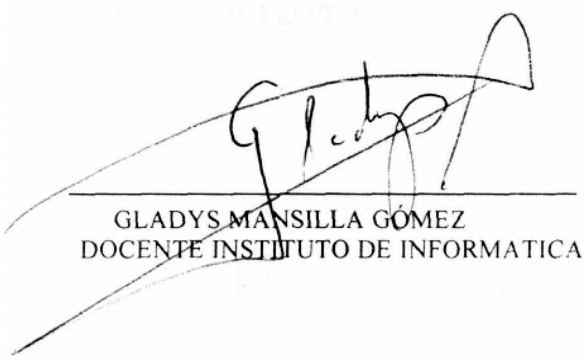
Nombre Trabajo de Titulación: "DESARROLLO DE UNA METODOLOGIA DE INTEGRACION DE SISTEMAS BASADO EN EL MONITOR TRANSACCIONAL BEA TUXEDO"

Nombre del alumno: KARINA ANDREA HERRERA HAASE
CLAUDIO JAVIER ALTAMIRANO ALTAMIRANO

Nota: 7.0 (en números) siete (en palabras)

Fundamento de la nota:

- En la realización de este trabajo de titulación se alcanzan plenamente los objetivos planteados al inicio.
- La presentación y redacción del informe están bien elaboradas, abarcando tópicos que inciden directamente en esta tesis y expresado en un lenguaje formal apropiado.
- El software desarrollado abarca todos los tópicos que requiere el tema, y la presentación es adecuada.
- Los alumnos han podido introducirse en la temática del monitoreo de transacciones y han sido capaces de desarrollar una aplicación.
- La metodología de integración que desarrollan, constituye un aporte de utilidad en la empresa del área a la que se abocan.



GLADYS MANSILLA GÓMEZ
DOCENTE INSTITUTO DE INFORMATICA

SANTIAGO, 01 DE JULIO DEL 2003

DE: PATRICIO E. VERA ANDRADE.
A : DIRECTORA DE ESCUELA INGENIERIA CIVIL EN INFORMATICA

MOTIVO

INFORME TRABAJO DE TITULACION

Nombre Trabajo de Titulación: "DESARROLLO DE UNA METODOLOGIA DE INTEGRACION DE SISTEMAS BASADO EN EL MONITOR TRANSACCIONAL BEA TUXEDO"

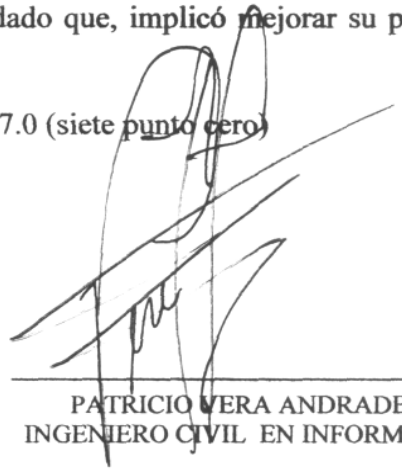
Nombre de los alumnos: KARINA ANDREA HERRERA HAASE
CLAUDIO JAVIER ALTAMIRANO ALTAMIRANO.

Nota: 7.0 (en números) siete (en palabras)

Fundamento de la nota:

La tesis esta bien lograda, puesto que explica en forma detallada la lógica de negocios que se tiene que implementar para que el monitor transaccional Tuxedo funcione adecuadamente en un ambiente empresarial. Por lo cual, los métodos aplicados, así como también, los criterios de análisis y desarrollo que se utilizaron en esta tesis, sustentan un trabajo que está rigurosamente elaborado y documentado. Además, la implementación exitosa que se realizó en BellSouth ratifica lo anteriormente expuesto, dado que, implicó mejorar su productividad y servicio al cliente.

Por esta razón, se califica con nota 7.0 (siete punto cero)



PATRICIO VERA ANDRADE
INGENIERO CIVIL EN INFORMÁTICA

Valdivia, 08 de Julio de 2003

De : Martín Gonzalo Solar Monsalves

A : Directora Escuela Ingeniería Civil en Informática

Ref. : Informe Calificación Trabajo de Titulación

Nombre Trabajo de Titulación:

"DESARROLLO DE UNA METODOLOGIA DE INTEGRACION DE SISTEMAS BASADO EN EL MONITOR TRANSACCIONAL BEATUXEDO".

Nombre Alumnos:

Karina Herrera H. - Claudio Altamirano A.

Evaluación:

Cumplimiento del objetivo propuesto	7.0
Satisfacción de alguna necesidad	7.0
Aplicación del método científico	6.5
Interpretación de los datos y obtención de conclusiones	6.0
Originalidad	5.0
Aplicación de criterios de análisis y diseño	6.5
Perspectivas del trabajo	6.0
Coherencia y rigurosidad lógica	6.5
Precisión del lenguaje técnico en la exposición, composición, redacción e ilustración	6.5
Nota Final	**6.3**

Sin otro particular, atte.:

Martín Solar Monsalves

INDICE

RESUMEN..... **I**
SUMMARY..... **II**
INTRODUCCIÓN..... **III**
OBJETIVOS DEL TRABAJO DE TITULACIÓN..... **V**

1 INTEGRACION DE APLICACIONES 1-1
1.1 Definición 1-1
1.2 Tipos de EAI..... 1-2
1.2.1 EAI a Nivel de Datos 1-2
1.2.1.1 Elementos del Diseño de EAI a Nivel de Datos 1-3
1.2.1.2 Elementos de Implementación de EAI a Nivel de Datos 1-5
1.2.1.3 Recomendaciones de Uso 1-7
1.2.1.4 Ventajas y Desventajas..... 1-7
1.2.2 EAI a Nivel de Interfaz 1-8
1.2.2.1 Integración a Nivel de Interfaz de Aplicación 1-8
1.2.2.2 Integración a Nivel de Interfaz de Usuario 1-10
1.2.2.3 Recomendaciones de Uso 1-11
1.2.2.4 Ventajas y Desventajas..... 1-11
1.2.3 EAI a Nivel de Método 1-12
1.2.3.1 Recomendaciones de Uso 1-14
1.2.3.2 Ventajas y Desventajas..... 1-15
2 DEFINICIÓN DE MIDDLEWARE 2-16
2.1 Middleware 2-16
2.2 Modelos de Middleware 2-20
2.2.1 Middleware Lógico Punto a Punto 2-21
2.2.2 Middleware Lógico Muchos a Muchos..... 2-21
2.2.3 Mecanismos de Comunicación Sincrónica 2-22
2.2.3.1 Requerimiento / Respuesta (Request / Reply) 2-23
2.2.3.2 Unidireccional 2-23
2.2.3.3 Polling 2-24
2.2.4 Mecanismos de Comunicación Asincrónica 2-25
2.2.4.1 Intercambio de Mensajes 2-26
2.2.4.2 Publicar / Subscribir (Publish / Subscribe) 2-26
2.2.4.3 Broadcast..... 2-27
2.3 Clasificación de Middleware..... 2-28

2.3.1 Middleware Orientado a Mensajes (MOM) 2-28
2.3.1.1 Conceptos 2-28
2.3.1.2 Colas de Mensajes 2-29
2.3.2 Llamada a Procedimientos Remotos(RPC) 2-31
2.3.2.1 Conceptos 2-31
2.3.2.2 Modelo RPC 2-32
2.3.2.3 Implementación RPC 2-34
2.3.2.4 Variaciones en el Modelo Requerimiento/Respuesta 2-34
2.3.2.5 Portabilidad e Interoperabilidad de Aplicaciones RPC 2-35
2.3.3 Arquitectura de Objetos Distribuidos 2-36
2.3.3.1 Conceptos 2-36
2.3.3.2 Caracteristicas de CORBA 2-38
2.3.3.3 Introducción a CORBA 2-40
2.3.4 Monitores de Procesamiento Transaccional 2-48
2.3.4.1 Conceptos 2-48
2.3.4.2 Modelo de un Monitor TP 2-51
2.3.4.3 Administración de Procesos de un Monitor TP 2-55
2.3.4.4 Recuperación y Administración de Sistema 2-57
2.3.4.5 Arquitectura de Monitores TP 2-58
3 DESARROLLO DE LA METODOLOGÍA 3-63
3.1 Definición de Tuxedo 3-63
3.1.1 Características 3-64
3.2 Componentes de Tuxedo 3-65
3.3 Tipos de Mensajes Tuxedo 3-70
3.3.1 Tipo STRING 3-70
3.3.2 Tipo FML y FML32 3-71
3.3.2.1 Especificación de la Tabla de Fields FML32 3-72
3.3.2.2 Generación del Archivo de Cabecera 3-73
3.3.2.3 Uso de los Archivos Generados 3-74
3.3.2.4 Tipos de datos e Identificación de Field 3-74
3.3.3 Tipo VIEW y VIEW32 3-76
3.3.3.1 Especificación del Archivo de texto VIEW (viewfile) 3-77
3.3.3.2 Compilación del Archivo de texto VIEW (viewfile) 3-78
3.3.3.3 Uso de los Archivos Generados 3-80
3.4 Clientes Tuxedo 3-80

3.4.1 Interfaz de Aplicación para el Control de Transacciones 3-80
3.4.2 Clientes Nativos 3-82
3.4.3 Clientes /WS 3-82
3.4.4 Estructura de Clientes 3-83
3.5 Servidores de Aplicación Tuxedo 3-85
3.5.1 Servicios Tuxedo 3-85
3.5.2 Estructura de Servidores 3-86
3.6 Seguridad en Tuxedo 3-87
3.6.1 Seguridad a Nivel de Servidor 3-87
3.6.2 Seguridad a Nivel de Cliente 3-89
3.7 Utilitarios WUD32/UD32 de TUXEDO 3-90
3.7.1 Formato del Archivo de Entrada 3-91
3.7.2 Ambiente UD32 3-92
3.7.3 Ambiente WUD32 3-93
3.8 Pruebas de Esfuerzo 3-94
3.9 Análisis del Tiempo de respuesta 3-95
3.10 Elementos de la Metodología 3-96
3.10.1 Estrategia IT de la Empresa 3-97
3.10.2 Arquitectura de la Empresa. 3-97
3.10.2.1 Desarrollo de una Política de Seguridad 3-97
3.10.2.2 Analizar los Requerimientos de la Empresa 3-99
3.10.2.3 Analizar los Requerimientos de Infraestructura 3-99
3.10.2.4 Evaluación de Aplicaciones 3-100
3.10.2.5 Especificación de la Arquitectura IT de la Empresa. 3-100
3.10.3 Arquitectura de la Aplicación 3-101
3.10.4 Desarrollo de Servicios 3-101
3.10.5 Integración y distribución de la Aplicación. 3-101
4 DESARROLLO DE LA INTEGRACIÓN DE APLICACIONES 4-103
4.1 Situación Actual 4-103
4.2 Principales Problemas 4-104
4.3 Evaluación de BEA Tuxedo 4-104
4.4 Arquitectura Tecnológica de Bellsouth sin Tuxedo 4-106
4.5 Diseño 4-107
4.5.1 Consideraciones Básicas 4-107
4.5.2 Protocolo de Mensajería 4-108

4.5.3 Adaptador de Integración con SAP 4-111
4.5.4 Arquitectura de Integración con Tuxedo 4-112
4.6 Implementación 4-113
4.6.1 Clientes Tuxedo 4-113
4.6.2 Servidores Tuxedo 4-120
5 CONCLUSIONES 5-125

ÍNDICE DE FIGURAS

1.2-1	Integración a Nivel de Datos	1-3
1.2-2	Integración a Nivel de Interfaz de Aplicación	1-9
1.2-3	Integración a Nivel de Interfaz de Usuario	1-10
1.2-4	Integración a Nivel de Métodos	1-13
2.2-1	Concepto de Middleware	2-17
2.1-2	Concepto de Middleware en relación a un Modelo de Referencia OSI y a un Modelo de Comunicaciones	2-20
2.2-1	Comunicación Sincrónica en Requerimiento / Respuesta	2-23
2.2-2	Comunicación Sincrónica Unidireccional	2-24
2.2-3	Comunicación Sincrónica de Polling	2-25
2.2-4	Intercambio de mensajes asincrónicos	2-26
2.2-5	Mecanismo de Comunicación Asincrónica Publicar / Subscribir	2-27
2.2-6	Modelo broadcast de comunicación asincrónica	2-28
2.3-1	Modelo de Middleware Orientado a Mensaje (MOM)	2-31
2.3-2	Modelo de Middleware RPC	2-33
2.3-3	Estructura del ORB de CORBA	2-43
2.3-4	Modelo de Middleware de un Monitor de Procesamiento Transaccional	2-52
2.3-5	Modelo de proceso por cliente	2-58
2.3-6	Modelo de único proceso	2-59
2.3-7	Modelo de muchos servidores y un único router	2-59
2.3-8	Modelo de muchos servidores y muchos routers	2-60
3.1-1	Concepto de BEA Tuxedo	3-63
3.2-1	Elementos de una Aplicación Tuxedo	3-67
3.4-1	Tipos de Clientes Tuxedo	3-80
3.4-2	Cliente /WS	3-83
3.4-3	Estructura de un Cliente Tuxedo	3-84
3.5-1	Estructura de un Servidor Tuxedo	3-87
3.7-1	Utilitarios WUD32 y UD32 de Tuxedo	3-91
4.4-1	Arquitectura de las Aplicaciones sin Tuxedo	4-103
4.5-1	Intercambio de mensajes FML32	4-108
4.5-2	Arquitectura General de Integración	4-109
4.5-3	Arquitectura Específica de Integración	4-110
4.6-1	Componentes del Cliente Centura	4-112
4.6-2	Arquitectura de Integración con AcuCobol	4-116
4.6-3	Secuencia lógica de los servidores	4-117

RESUMEN

El proyecto de tesis que aquí se presenta define una metodología de integración de sistemas basado en el monitor transaccional Tuxedo. Se describe en términos generales y puede ser aplicada a cualquier empresa que tenga problemas de integración.

El contenido de la tesis considera los siguientes puntos.

- Presentación conceptual de la integración de sistemas y de los diversos niveles de integración que se pueden implementar.
- Definición de middleware y descripción de los distintos tipos de middleware existentes: RPC, MOM, Objetos Distribuidos y Monitores de Procesamiento Transaccional.
- Definición de Tuxedo y descripción de sus elementos, entre ellos: Cliente y Servidores Tuxedo, Servicios, Dominios y Tipos de Mensajes.
- Descripción de la metodología.
- Desarrollo de la integración de aplicaciones en Bellsouth. Considera el diseño e implementación de las aplicaciones y de la arquitectura de la solución.

SUMMARY

The thesis project that presented here defines a methodology of integration of systems based on the transactional monitor Tuxedo. It is described in general terms and it can be applied to any company that has integration problems.

The content of the thesis concentrate on the following points:

- Conceptual presentation of Systems Integration and each type of integration that can be implemented.
- Middleware definition and description of the different types of existent middleware: RPC, MOM, Distributed Objects and Monitors of Transactional Processing.
- Definition of Tuxedo and description of it's components, between them: Client and Servers Tuxedo, Services, Domains y Type Buffers.
- Description of methodology used.
- Development of applications integration in Bellsouth. Considering the design and implementation of the applications and architect of the solution.

INTRODUCCIÓN

El panorama de la empresa actual está compuesto de un conjunto de sistemas que son el resultado de la evolución de los negocios y de la tecnología de los últimos años. Los sistemas han sido desarrollados para solucionar problemas específicos de alta prioridad o para realizar mejoramientos de productividad. Al mismo tiempo, se han seguido distintos caminos para lograr una integración básica entre los sistemas, esencialmente con el objeto de evitar procesos manuales o para el mejoramiento productivo. En la mayoría de los casos, se han utilizados bases de datos como medio de intercambio de información. Sin embargo, esa información no está en un solo lugar y por lo general produce inconsistencias y duplicidad de la información.

La evolución de la tecnología también ha contribuido a la fragmentación y caos de los sistemas. En la medida que se van desarrollando nuevos sistemas, se opta por la última tecnología para su implementación. De este modo, los antiguos sistemas ya no pueden ser implementados con esta nueva tecnología porque hay implícito un trabajo de desarrollo que no es menor. Por lo tanto, si los antiguos sistemas funcionan como corresponde, no hay grandes incentivos en la empresa para migrarlos a una nueva tecnología. En base al razonamiento anterior, con el paso del tiempo los antiguos sistemas van siendo cada vez más difíciles de integrar y de operar, porque avanza la tecnología y los procesos de negocio de la empresa solo se van incorporando a los nuevos sistemas.

En las grandes organizaciones, es natural la existencia de islas tecnológicas y organizacionales que han nacido como resultado de un avance en la especialización funcional de sus departamentos en el tiempo. Estos han sido incapaces de lograr compartir ideas y de encontrar soluciones comunes al tema del desarrollo de sistemas. Algunas grandes empresas, habiendo tomado conciencia de esto, han optado por comprar soluciones empaquetadas, que si bien integran gran parte de la empresa, no son capaces de cubrir el 100% de los aspectos del negocio de la empresa. Como consecuencia, se crea un problema adicional, la dificultad inherente de integrarse con este tipo de soluciones.

Debido a lo anterior, surge la necesidad de eliminar las barreras internas de una empresa a través de una integración con una visión global de la empresa.
En la actualidad el éxito de un negocio está directamente relacionado con la velocidad a la cual puede responder a nuevos modelos de negocio y a cambios en sus mercados destino y a la manera en como utiliza la tecnología de información para entregar sus productos y servicios.

Las organizaciones de negocio están evolucionando continuamente. Sin embargo, para ser exitoso deben evolucionar más rápido y más efectivamente que sus competidores.
En base a lo expuesto anteriormente, esta tesis contempla los siguientes temas divididos en cuatro capítulos que se describen a continuación:
En el capítulo uno se describe el problema de integración y se describen además los niveles en los cuales se pueden integrar las aplicaciones de una empresa: Integración a nivel de datos, a nivel de método, a nivel de interfaz de usuario y a nivel de interfaz de aplicación.
En el capítulo dos, se define el concepto de middleware y se detallan los tipos de middleware existentes: Objetos distribuidos, RPC, MOM y monitores de procesamiento transaccional.
En el capítulo tres se define lo que es Tuxedo y luego se detallan los elementos que lo componen: Clientes, Servidores, Dominios. Adicionalmente se describe la metodología.
En el capítulo cuatro se detalla en forma específica la arquitectura de solución planteada en Bellsouth y los sistemas que fueron integrados. Para cada caso se define la arquitectura de la aplicación. Este capítulo termina con las conclusiones que se obtuvieron en el trabajo realizado.
Todo lo expuesto anteriormente abarca el trabajo de titulación denominado: “Desarrollo de una Metodología de Integración de Sistemas basado en el Monitor Transaccional Tuxedo”

OBJETIVOS DEL TRABAJO DE TITULACIÓN

Objetivos Generales

Presentar una metodología que permita la integración de sistemas existentes al interior de una empresa a través del uso del monitor transaccional Tuxedo.

Objetivos Específicos

Definir los conceptos teóricos del monitor transaccional Tuxedo y sus potencialidades.

Definir los elementos técnicos necesarios que permitan realizar la integración entre uno o más dominios Tuxedo y los sistemas existentes.

Definir los pasos que permitan establecer en base a la funcionalidad deseada los requerimientos de integración específicos de cada sistema involucrado.

Desarrollar la integración de sistemas existentes en distintas plataformas utilizando la metodología propuesta.

Establecer mecanismos para realizar pruebas de esfuerzo de los Servidores de Aplicación Tuxedo.

Establecer el procedimiento que permita medir tiempos de respuestas de los Servidores de Aplicación Tuxedo.

1 INTEGRACION DE APLICACIONES

1.1 DEFINICIÓN

La **Integración de Aplicaciones Empresariales** (EAI) permite a una organización establecer una infraestructura tecnológica que une de manera transparente las aplicaciones de negocio heterogéneas – tanto empaquetadas como implementadas en casa – en un sistema unificado, tal que los procesos y datos puedan ser compartidos a través de la compañía y más allá, para incluir clientes y socios comerciales [1].

Debemos pensar que el estado actual de las Tecnologías de Información(IT) de las grandes empresas ha sido el resultado por un lado de haber implementado internamente sistemas que resuelven la problemática de sus departamentos y por otro, de haber comprado sistemas empaquetados de clase mundial. De hecho, no es difícil encontrar empresas con sistemas basados en el modelo Cliente/Servidor con su respectivo Administrador de Base de Datos Relacional (Sybase, Oracle, SQL Server, etc.), una solución clásica en la década pasada. Sin embargo, construirlo todo no ha sido la única alternativa, hay empresas que optaron por comprar herramientas de clase mundial que implementan gran parte de los procesos de negocio de la empresa. En tal sentido, es importante entender conceptos relacionados con este tipo de productos.

La primera de ellas corresponde a las aplicaciones de **Planificación de Recursos Empresariales** (ERP) están constituidas por diversos módulos interrelacionados entre sí, de manera tal que logran la integración de la empresa abarcando diferentes áreas internas de una organización.

Estas soluciones, que nacieron como respuesta a las necesidades de información financiera en las empresas, paulatinamente han incorporado también funcionalidades de las áreas logísticas (Ventas, Producción, Gestión de Materiales, Mantenimiento, etc.), Gestión de RR.HH. y últimamente podemos observar como incluyen dentro de su estándar las más novedosas tecnologías (Internet, Workflow, Gestión Documental, etc.) y soluciones específicas de negocio.

Dentro de las principales soluciones de este tipo están: R/3 de SAP, Baan ERP, Peoplesoft, J. D. Edwards y Oracle.

Por otro lado, la definición más aceptada en lo que se refiere a soluciones de **Gestión de la Relación con Clientes** (CRM), es la que lo describe como el conjunto de estrategias de ventas, marketing, comunicación y tecnologías diseñadas con el propósito de establecer relaciones duraderas con todos los clientes, identificando y satisfaciendo sus necesidades.

CRM es una visión integral de la empresa sobre cómo debe relacionarse con los clientes, cuál es el canal que debe emplear, la herramienta tecnológica que debe utilizar para poder tener un trato masivo y simultáneo con cientos o miles de sus clientes. Asimismo el CRM balancea la organización empresarial hacia el cliente: cambia el foco desde la "operación" para centrarse en la figura del comprador de sus productos y servicios.

Dentro de las principales soluciones de este tipo están: PeopleSoft CRM, mySAP CRM, Siebel.

Un CRM, como se mencionó anteriormente, se encarga de la administración de la relación con los clientes y con lo que respecta a su relación con el ERP, estos son dos modelos de sistemas complementarios, pero distintos. Mientras que el ERP se dedica al back office (operaciones internas de una empresa), el CRM se enfoca al front office (los clientes que forman parte del exterior de la empresa). Los ERP y los CRM trabajan de manera conjunta para generar una oferta integral, logrando que las empresas se incorporen por completo al negocio electrónico.

1.2 TIPOS DE EAI

1.2.1 EAI a Nivel de Datos

El modelo de integración de datos permite la integración de software a través del acceso a los datos que son creados, manipulados y almacenados por un software con el objeto de reutilizarlos y sincronizarlos a través de múltiples aplicaciones. Este modelo accede directamente a las bases de datos u otras fuentes de datos ignorando la capa de presentación y la capa de lógica de negocios para crear la integración. La integración a nivel de datos, mostrada en la figura 2.2-1, puede ser tan simple como el acceso a sistemas de administración de bases de datos relacionales o tan complejo como manejar

bases de datos de productos empaquetados, o algún sistema de archivos propietarios de una aplicación.

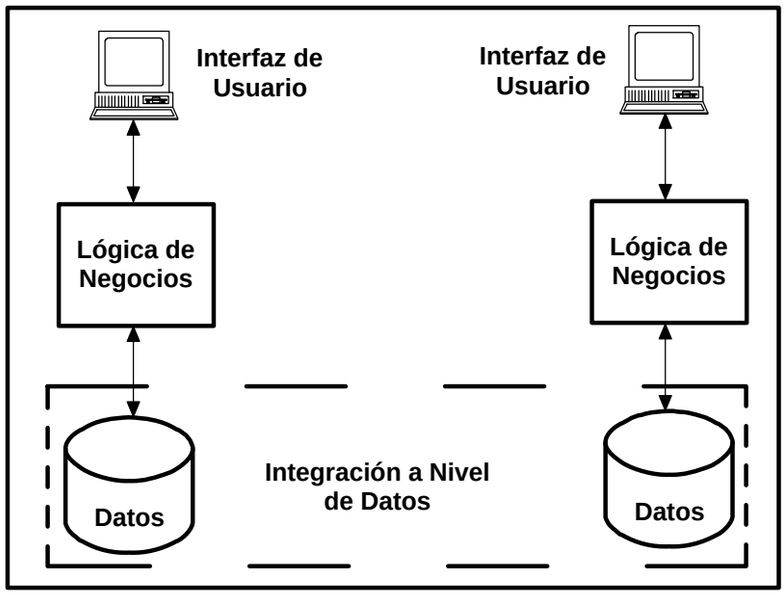


Figura 1.2-1. Integración a Nivel de Datos.

A continuación presentaremos una serie de conceptos que son necesarios para poder comprender la integración a nivel de datos:

Metadato se entiende como datos acerca de otros datos. Ejemplo:

- Un catálogo de librería contiene información (metadato) acerca de publicaciones (dato).
- Un sistema de archivos mantiene permisos (metadato) acerca de archivos (dato).

En el contexto que nos interesa, nuestros metadatos incluirán fuentes de datos, tipos, restricciones y derechos de acceso.

1.2.1.1 Elementos del Diseño de EAI a Nivel de Datos

La implementación de integración a nivel de datos requiere conocer la ubicación de los datos, recolectar información a cerca de los datos y aplicar principios del negocio para determinar el flujo de datos exactos que se debe implementar.

Identificación de los Datos

La primera tarea en el proceso de identificación de los datos es crear una lista de sistemas que se necesiten integrar. Basados en esta lista, es posible determinar las fuentes de datos (bases de datos, archivos, etc.) que dan soporte a estos sistemas. Por cada sistema, se debe definir la base de datos principal. La descripción de cada base de datos debería incluir la localización física, modelo y una revisión del esquema y tecnología de la base de datos. Cualquier tecnología que sea capaz de hacer ingeniería de reversa permitirá obtener esquemas físicos de la base de datos, los cuales facilitarán la identificación de los datos dentro del dominio del problema. Además, se debe documentar la forma en que las aplicaciones usan esta data, incluyendo reglas específicas a nivel de sintaxis como de semántica. Esto es necesario para que la solución de integración mantenga la integridad de los datos llevada a cabo por la aplicación. El formato de los datos es otro componente importante que se debe considerar. Esto permite determinar cómo está estructurada la información, incluyendo propiedades del elemento dato, dentro de la estructura. Diferentes estructuras y esquemas pueden necesitar una comprensión de los formatos de datos para que las estructuras y esquemas sean transformados en la medida que la información se mueve de un sistema a otro. Finalmente, información acerca de la latencia de los datos, es decir, determinar cuan necesario es que la información esté, es otra propiedad de los datos que necesita ser determinada.

Catalogación de los Datos

Para la integración a nivel de datos, la catalogación de datos es el proceso de recolectar metadatos y otros datos en el contexto del dominio del problema. Una vez realizado esto, se puede crear un catálogo de todos los elementos del dato en toda la empresa. Esta es la base del entendimiento necesario para crear el modelo de metadatos empresarial que es, la base de la integración a nivel de datos, lo cual requiere una comprensión total del diccionario de datos. Un diccionario de datos incluye la información tradicional de este y toda la información de interés para el proceso de integración. Esto incluye la información de sistema, seguridad, propiedad, procesos conectados, comunicaciones y aspectos de integridad, en conjunto con metadatos tradicional como formato, nombre de atributo y descripción.

Construcción del Modelo de Metadatos Tradicional

Este modelo se usará como una guía maestra para la integración de diversas fuentes de datos. El modelo de metadatos define todas las estructuras de datos y la manera en que estas interactúan dentro del dominio de la solución. El catálogo de datos define los parámetros del problema que el modelo de metadatos soluciona. Una vez construido el modelo, este se constituye en el repositorio de la empresa y en el directorio maestro para la solución de integración. El repositorio puede solucionar el problema de integración a nivel de datos y adicionalmente proveer una base de trabajo para soluciones más robustas en el futuro.

1.2.1.2 Elementos de Implementación de EAI a Nivel de Datos

Base de Datos - Base de Datos

Este esquema de integración es algo que se ha realizado por años. La integración Base de Datos - Base de Datos (DB-DB) significa en términos simples compartir información a nivel de base de datos y, de esta manera, lograr la integración de aplicaciones. La integración DB-DB puede existir en configuraciones del tipo una a una, una a muchas y muchas a muchas. El concepto DB-DB se puede aproximar con el tradicional middleware de bases de datos y con software de replicación de bases de datos, características comunes en los principales motores de bases de datos relacionales de hoy en día (Sybase, Oracle). Por otro lado, los brokers de mensajes también trabajan con integración DB-DB, pero ante la imposibilidad de compartir métodos coherentemente o la necesidad de acceder a sistemas complejos (aplicaciones ERP) ellos se ven sobrepasados.

Hay dos tipos de soluciones en el contexto de la integración DB-DB. La primera es la **replicación** básica que mueve información entre bases de datos que mantienen el mismo esquema de información sobre todas las bases de datos de origen y destino. La segunda solución es la **replicación y transformación**. Al utilizar este tipo de productos, es posible mover información entre diferentes tipos de bases de datos, incluyendo diversas marcas (Sybase, Oracle, Informix) y modelos (relacional, orientada a objetos

y multidimensional), transformando los datos en el instante de manera tal que sean representados correctamente en las bases de datos destinos.

La ventaja de este modelo de integración es la simplicidad. Al tratar con la información de la aplicación a nivel de los datos, en general no hay necesidad de cambiar la aplicación de origen o la aplicación de destino. Esto reduce el riesgo y costo de implementación de la integración de aplicaciones en una empresa. Por último, se debe indicar que hay aplicaciones en que la lógica de la aplicación está ligada a los datos y, de esta manera es difícil manipular la base de datos sin modificar la lógica de la aplicación o, al menos, la interfaz de la aplicación. Esto es muy común en el caso de SAP R/3, donde para evitar problemas de integridad, actualizar la base de datos generalmente demanda usar la interfaz (RFCs y BAPIs) definida por SAP R/3.

Base de Datos Confederadas

La integración de Bases de Datos Confederadas también trabaja a nivel de bases de datos, como la integración DB-DB. Sin embargo, en lugar de simplemente replicar los datos a través de diversas bases de datos, el software de Bases de Datos Confederadas permite al desarrollador acceder a cualquier número de bases de datos, usando diversas marcas, modelos y esquemas, a través de un solo modelo de base de datos virtual. Este modelo de base de datos virtual existe sólo en software y está mapeado a cualquier número de bases de datos físicas conectadas. El desarrollador utiliza esta base de datos virtual como un solo punto de integración, accediendo a datos de diversos sistemas a través de la misma interfaz de base de datos.

La ventaja de este método es la seguridad sobre el middleware para compartir información entre aplicaciones y no una solución personalizada. Además, el middleware oculta las diferencias en las bases de datos integradas de otras aplicaciones que están usando la visión integrada de las bases de datos. Desafortunadamente, este no es un verdadero método de integración; a pesar de haber una visión de varias bases de datos en un "modelo unificado", existirá aun la necesidad de crear la lógica para la integración de las aplicaciones con las bases de datos.

1.2.1.3 Recomendaciones de Uso

Se recomienda usar integración a nivel de datos en los siguientes casos:

- Cuando se desee combinar datos de múltiples fuentes para análisis y toma de decisiones.
- Permitir que diversas aplicaciones puedan leer los datos de una fuente de información común. Por ejemplo, cuando se desee crear un sistema de data warehouse que tiene información completa de los clientes y que pueda ser accedida por una variedad de aplicaciones estadísticas y de data mining.
- Permitir que los datos puedan ser extraídos de una fuente y reformateados y actualizados en otra. Por ejemplo, cuando se desee actualizar la información relacionada con la dirección del cliente en todas las fuentes de datos tal que ellas permanezcan sincronizadas y consistentes.

1.2.1.4 Ventajas y Desventajas.

Este modelo de integración otorga un mayor grado de flexibilidad que la integración a nivel de presentación. Provee acceso a un rango de datos más amplio que cuando se integra a nivel de interfaz de usuario. Este método también simplifica el acceso a las fuentes de datos. Cuando las bases de datos proveen interfaces de fácil acceso o cuando existe un middleware que integra múltiples fuentes de datos a nuevas aplicaciones, de esta manera este modelo permite simplificar la integración.

El modelo de integración de datos también permite reutilizar los datos a través de otras aplicaciones, es decir, una vez que la integración se ha completado nuevas aplicaciones pueden hacer uso de esta información.

La necesidad de reescribir la lógica de negocios puede parecer un problema menor, pero en la realidad puede transformarse en un problema muy complejo. Por ejemplo, consideremos un banco donde se ha utilizado la integración a nivel de datos para acceder la información de una cuenta corriente. La lógica para calcular el saldo de una cuenta podría ya existir en la lógica de negocio de la aplicación que crea y usa la base de datos, pero podría no estar disponible a otras aplicaciones que fueron integradas usando el modelo de integración de datos. En tal situación habría que escribir la

lógica para recalcular el saldo de la cuenta corriente. Esto podría suceder porque el chequeo del saldo de la cuenta podría no reflejar los depósitos de ese momento. Esta lógica duplicada podría tener un alto costo en la creación y mantención. Por último, si la integración está basada en un modelo de datos y, este cambia, la integración puede verse afectada, haciendo de este modo la integración sensible a los cambios. Esto es por una cuestión natural debido a la naturaleza evolutiva de los sistemas, lo que conduce a un esfuerzo significante en mantener la integración.

1.2.2 EAI a Nivel de Interfaz

1.2.2.1 Integración a Nivel de Interfaz de Aplicación

Este modelo está formado por interfaces que los desarrolladores exponen de una aplicación empaquetada o una aplicación personalizada para tener acceso a diversos niveles o servicios de una aplicación. Algunas veces dichas interfaces permiten el acceso a procesos de negocios, algunas veces permiten el acceso directo a los datos y otras a ambos.

Los desarrolladores exponen esas interfaces por dos razones. La primera es la de proveer acceso a procesos de negocios y datos encapsulados dentro de las aplicaciones que ellos han creado sin forzar a otros desarrolladores a llamar a la interfaz de usuario (definida en el punto 1.2.2.2) o ir directamente a la base de datos. Tales interfaces permiten a aplicaciones externas acceder a los servicios de esos paquetes o aplicaciones personalizadas sin hacer cambios a los paquetes o a las aplicaciones en sí. La segunda razón para exponer estos tipos de interfaces es la de proveer un mecanismo que permita compartir informaciones encapsuladas. Por ejemplo, si los datos de SAP se requieren desde Excel, SAP expone las interfaces que permiten al usuario invocar un proceso de negocios o recolectar datos comunes.

Las interfaces de aplicación constituyen un tipo distinto de EAI debido a que los escenarios indicados en el párrafo anterior son distintos a la integración a nivel de método (definida en el punto 1.2.3) o a la integración a nivel de interfaz de usuario. Mientras es posible distribuir los métodos que existen dentro de las empresas entre diversas aplicaciones, normalmente se comparten usando un mecanismo de lógica de procesos de negocio común, tal como un servidor de aplicaciones o un objeto distribuido. La integración a nivel de interfaz de usuario es similar a la integración a nivel de interfaz de

aplicación en que tanto los datos y los procesos de negocios quedan disponibles a través de una interfaz expuesta por las aplicaciones empaquetadas o personalizadas.

Las potenciales complejidades de las interfaces de aplicación, así como también la naturaleza dinámica de las interfaces en sí, aumenta la diferencia con la interfaz de usuario.

Debido a que las interfaces varían ampliamente en el número y calidad de las características y funciones que proveen, es casi imposible saber qué esperar cuando se invoca una interfaz de aplicación cuando esta no está documentada.

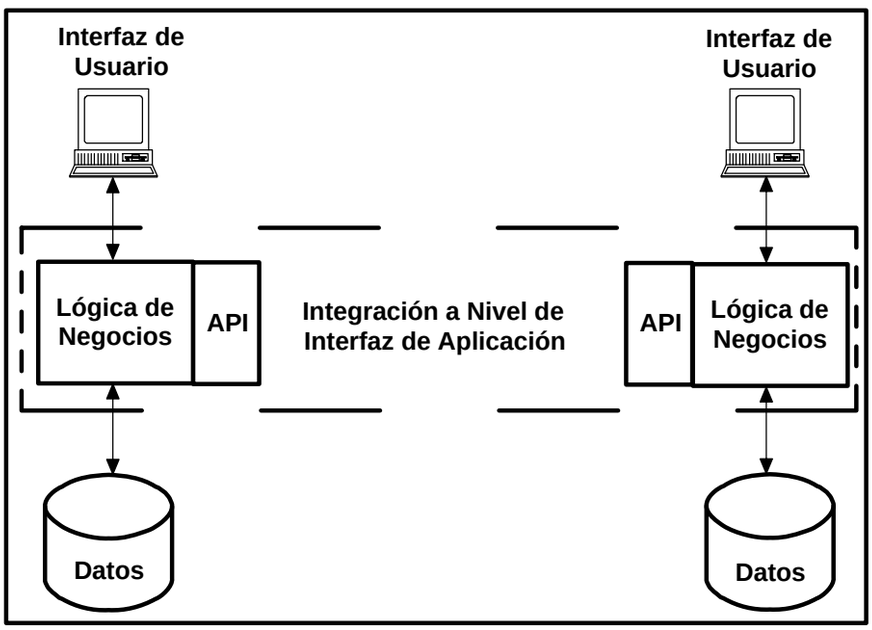


Figura 1.2-2. Integración a Nivel de Interfaz de Aplicación.

En el caso que nos interesa es importante hacer un análisis especial sobre SAP, porque constituye el paquete ERP más usado en las empresas. Está formado por una interfaz gráfica denominada SAP GUI y un servidor de aplicaciones que maneja su propia base de datos relacional. Una de las formas de acceder a la funcionalidad de SAP es a través de APIs exportadas y conocidas como **Llamada a Funciones Remotas** (RFC). De este modo, al llamar a estas RFCs se accede a la lógica de negocio existente en SAP manteniendo la integridad transaccional. Otro mecanismo de más bajo nivel es el de **BAPIS**, que permite acceder a los datos y procesos de SAP.

1.2.2.2 Integración a Nivel de Interfaz de Usuario

El modelo de integración a nivel de interfaz de usuario es una de las formas más simples de integración. En este modelo la integración de múltiples componentes de software es ejecutada a través de una interfaz de aplicación de usuario. Típicamente la integración resulta en una nueva presentación unificada para el usuario. La nueva presentación parece ser una única aplicación aunque pueda estar haciendo uso de diversas aplicaciones heredadas. La lógica de integración, las instrucciones a donde se dirigen las interacciones del usuario, comunican la interacción del usuario al software apropiado usando sus presentaciones existentes como un punto de integración. Entonces integra cualquier resultado generado desde los diversos componentes de software integrados. Por ejemplo, la herramienta de acceso a información de las pantallas a través de mecanismos programáticos podría ser usada para tomar un conjunto de aplicaciones mainframe e integrarlas en una nueva aplicación Microsoft Windows. Esta única presentación debería reemplazar un conjunto de interfaces basada en terminales y podría incorporar características adicionales, funciones, y flujos de trabajo para el usuario. Esto crea un mejor flujo entre la aplicaciones heredadas del usuario.

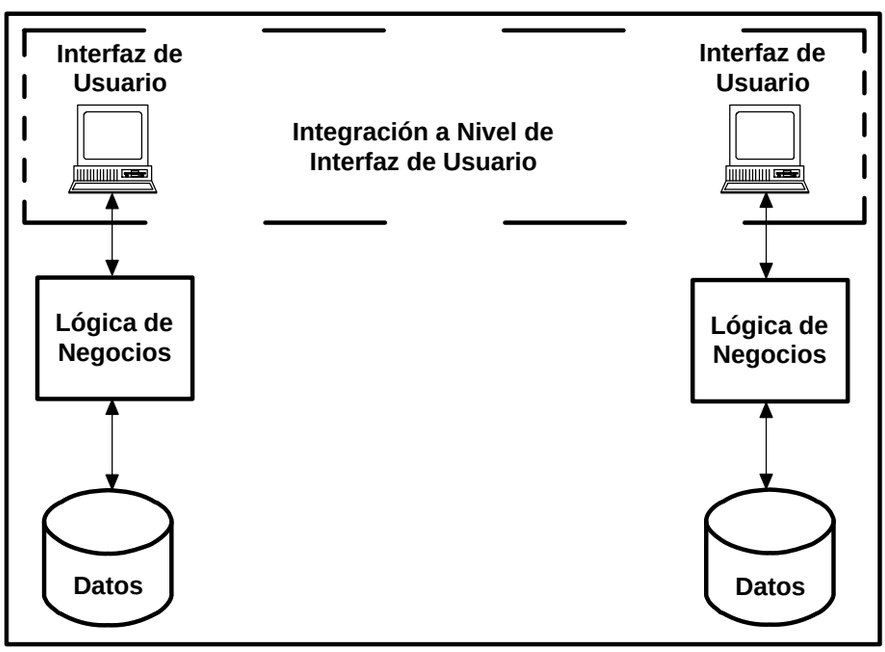


Figura 1.2-3. Integración a Nivel de Interfaz de Usuario.

1.2.2.3 Recomendaciones de Uso

Se recomienda usar integración a nivel de interfaz de usuario en los siguientes casos:

- Colocar una interfaz de usuario basada en un computador sobre una aplicación existente basada en terminales (VT-100) para proveer una aplicación fácil de usar para un usuario final.
- Presentar una interfaz que el usuario perciba como una sola aplicación, cuando en realidad se trata de varias aplicaciones.
- Integrar con una aplicación cuyo único punto de integración útil y viable es a través de su presentación.
- Esta forma de integración es útil sólo cuando la integración puede ser realizada usando la interfaz de usuario o presentación de las aplicaciones heredadas. La integración de este tipo está normalmente orientada a la interfaz de usuario de texto tales como IBM 3270 o Interfaces VT-100. Ejemplos donde el modelo de integración de presentación es más adecuada son:
  - Proveer una interfaz Microsoft Windows a una aplicación mainframe.
  - Proveer una interfaz HTML unificada a una aplicación mainframe o SAP R/3.
  - Proveer una interfaz unificada basada en Java a múltiples aplicaciones mainframes.
- Se recomienda utilizar integración a nivel de interfaz de aplicación cada vez que se necesite integrar con aplicaciones empaquetadas, siendo esta quizás, casi la única manera de poder integrar.

1.2.2.4 Ventajas y Desventajas

La integración de usuario es muy fácil de realizar y puede ser hecha con relativa rapidez. La lógica de presentación normalmente es menos compleja que la lógica funcional o de datos porque puede ser visualizada gráficamente, además normalmente está bien documentada y es autodescriptiva. Cuando las herramientas usadas para realizar esta integración trabajan bien ellas hacen la mayoría del trabajo necesario para crear la integración. El desarrollador se concentra sólo en la construcción de la nueva presentación.

Por el otro lado, la integración de presentación ocurre sólo a nivel de interfaz de usuario. Por lo tanto, sólo los datos y las interacciones definidas en las presentaciones heredadas pueden ser accedidas. Además, la integración de presentación puede tener cuellos de botella en el rendimiento porque agrega una capa de software extra a las aplicaciones ya existentes. La lógica y datos que conforman la base de las aplicaciones existentes no pueden ser accedidas.

El modelo de integración de presentación es el más limitado de los tres. La integración toma lugar en la presentación y no en la interconexión entre las aplicaciones y datos.

1.2.3 EAI a Nivel de Método

La integración a nivel de método permite que la empresa sea integrada a través de un conjunto de métodos o lógica de negocios compartida. Esto se lleva a cabo definiendo métodos que pueden ser compartidos y, por lo tanto integrados, por un conjunto de aplicaciones o proveyendo la infraestructura para lograr compartir los métodos. Los métodos pueden ser compartidos ya sea estén organizados sobre un servidor central o accediéndolos entre aplicaciones (ejemplo Objetos Distribuidos).

El intento de compartir procesos comunes tiene una larga historia comenzando hace más de diez años atrás con la tendencia de objetos distribuidos y con el modelo cliente / servidor multicapa. Un conjunto de servicios distribuidos sobre un servidor común que provee a la empresa de la infraestructura de reutilización e integración de lógica de negocios. En este contexto cabe destacar la importancia del concepto de **reutilización** definiéndolo como un conjunto de métodos comunes, capaces de reutilizar esos métodos entre las aplicaciones de la empresa. Como consecuencia, se ve reducida significativamente la duplicidad o redundancia de aplicaciones y métodos. No obstante lo anterior, la reutilización absoluta no ha sido implementada aún en las empresas. Las razones para esto pueden pasar por falta de políticas internas o la incapacidad de seleccionar un conjunto de tecnología consistente. En la mayoría de los casos, la limitante en la reutilización está basada en la falta de una arquitectura corporativa y de control central. Es por eso, que necesitamos de una metodología de integración basada en herramientas y técnicas que permitan crear en las empresas no sólo la oportunidad para aprender como compartir métodos

comunes, sino que también, la infraestructura necesaria para que dicha plataforma compartida sea una realidad. Mientras lo anterior suena como una solución de integración perfecta, debemos indicar que también constituye el método de integración más invasivo. A diferencia de los otros métodos de integración (a nivel de datos y a nivel interfaz), los cuales típicamente no requieren cambios a las aplicaciones, la integración a nivel de método requiere que varias, sino todas, las aplicaciones de la empresa sean modificadas para obtener las ventajas de este modelo.

Cambiar las aplicaciones es una proposición muy costosa. Además de cambiar la lógica de la aplicación, se debe proceder a probar la funcionalidad, integrar y redistribuir las aplicaciones dentro de la empresa, un proceso que normalmente causa costos que deben ser evaluados por la plana directiva.

Considerando la naturaleza invasiva y costosa de la integración a nivel de método, las empresas deben entender claramente el valor de sus oportunidades y riesgos en el proceso de evaluación. El ser capaces de compartir lógica de negocios que es común a varias aplicaciones e integrar dichas aplicaciones, constituye una tremenda oportunidad. Sin embargo, existe el riesgo implícito asociado al costo de implementación de la integración a nivel de método que puede opacar las ventajas buscadas.

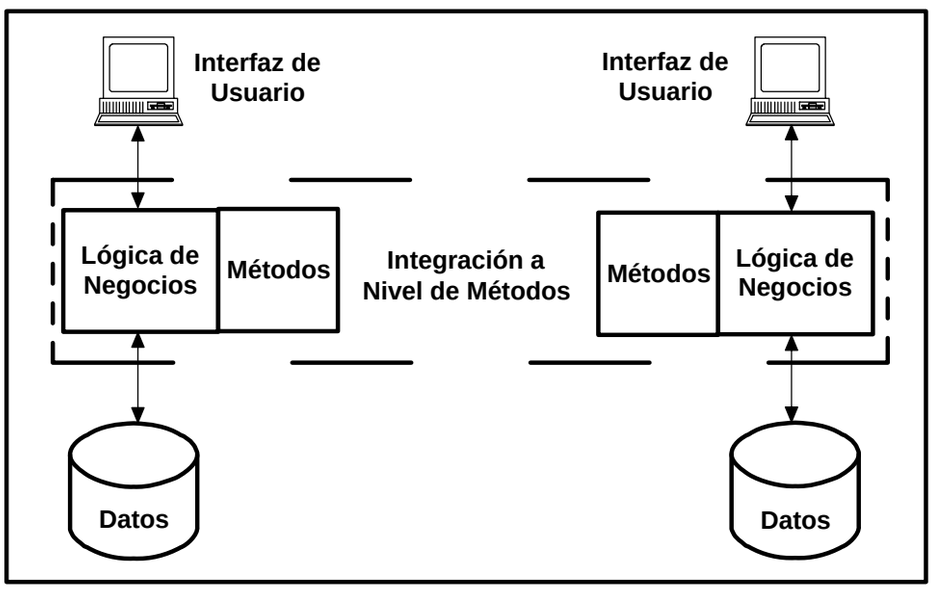


Figura 1.2-4. Integración a Nivel de Métodos.

1.2.3.1 Recomendaciones de Uso

El modelo de integración a nivel de método es único en el sentido que puede solucionar un amplio conjunto de problemas, incluyendo la solución de los mismos problemas que pudiesen resolverse a través del modelo de integración a nivel de datos y de interfaz. Esto se realiza interviniendo el código con la lógica del negocio. Por lo anterior, varias de las recomendaciones expuestas en la integración a nivel de datos también son aplicables en este modelo de integración.

Decidir cuando usar integración de método versus integración de datos o de interfaz para solucionar problemas depende de varios factores. El primero es la factibilidad de acceder a la funcionalidad de las aplicaciones. En algunos casos el acceso puede ser tan difícil que el único camino de integración sea a nivel de datos o a nivel de interfaz. El segundo factor está relacionado con el rendimiento. El método seleccionado debe cumplir con los requerimientos de rendimiento del sistema. El desempeño debe ser fijado en base a mediciones sobre casos de uso porque es dependiente de una situación en particular. Finalmente, se debe considerar lo importante que es la reutilización en el futuro de la empresa. La integración a nivel de método permite una mayor reutilización, pero es más difícil de aplicar, lo cual depende del diseño del código que realiza la función que debe ser accedida. Áreas adicionales donde la integración a nivel de método es más aplicable son:

- Cuando una nueva aplicación deba tomar una decisión, tal como hacer un depósito o colocar una orden de pedido que está manejada por otra aplicación o por un conjunto de ellas. Esto requiere la integración a nivel de método para poder procesar la solicitud.
- Al enfrentarse a un flujo de trabajo en la integración, tal como el procesamiento de una orden de pedido en que intervienen los canales de distribución y cargo en la cuenta. Esto necesita la integración a través de un conjunto de aplicaciones similares para el ejemplo previo, pero con la secuencialidad implementada en el software de integración.
- Cuando se debe garantizar la integridad transaccional entre aplicaciones. Por ejemplo, cuando se desee asegurar que el depósito sea terminado antes de entregar el recibo al cliente. Es importante considerar que este es uno de los tipos más difíciles.

- Cuando se desea anticipar un nivel de reutilización sustancial en la lógica de negocios. Esencialmente se trata de permitir que múltiples aplicaciones tengan acceso a la función de negocio sin necesidad de proceder a la implementación en cada una de aplicaciones involucradas. Este tipo de integración transforma aplicaciones existentes en elementos reutilizables.

1.2.3.2 Ventajas y Desventajas

El modelo de integración a nivel de método provee la capacidad más robusta de integración de los otros dos modelos. Es el más flexible en los problemas que puede resolver. Puede ser usado para resolver los problemas de integración a nivel de interfaz y de datos. Lo más importante, es que provee un alto grado de reutilización de los elementos que crean los otros dos modelos si son aplicados apropiadamente.

Este modelo, sin embargo, genera un aumento en las complejidades al intentar integrar al nivel de lógica de negocio. La curva de aprendizaje para este tipo de implementaciones es bastante más complejo que los anteriores, por cuanto, necesariamente se debe entender la lógica del negocio, la cual no siempre está documentada y, en este caso, se debe recurrir directamente al código de los sistemas en los casos que sea posible.

2 DEFINICIÓN DE MIDDLEWARE

2.1 MIDDLEWARE

El concepto de Middleware podría considerarse un tanto difuso [2], a menudo usado sin una referencia a una base teórica sólida. Esto se debe principalmente a la falta de publicaciones científicas en revistas técnicas. Sin embargo, cuando construimos aplicaciones estructuradas, en particular distribuidas, es difícil no enfrentarse a este concepto. Aunque la literatura acerca de middleware es relativamente escasa, podemos identificar las siguientes características que nos permitirán entender el concepto que hay detrás:

- El middleware es un servicio a nivel de aplicación. Implementa servicios para aplicaciones de alto nivel. La funcionalidad común es agrupada en una capa llamada middleware, permitiendo a los desarrolladores concentrarse en la funcionalidad específica de la aplicación.
- El middleware está fuertemente relacionado a los sistemas distribuidos. Dirigido esencialmente a una gama de aplicaciones complejas sobre varios sistemas, es decir, usado en un ambiente distribuido.
- El middleware se ejecuta sobre diferentes plataformas. El middleware necesita estar disponible sobre plataformas heterogéneas para soportar independencia entre el ambiente y la aplicación. Los ambientes más comúnmente usados para aplicaciones distribuidas están basados en el modelo cliente/servidor, donde los servidores son típicamente servidores UNIX o Windows NT y los clientes son computadores personales, Mac y hasta estaciones de trabajo UNIX en arquitecturas de tres capas.
- El middleware toma ventajas del sistema operativo o servicios del DBMS (Sistemas de Administración de Base de Datos).

Por lo tanto, el middleware apunta a reducir el impacto de problemas relacionados al desarrollo de aplicaciones dentro de ambientes heterogéneos, ofreciendo servicios estandarizados de alto nivel que ocultan la mayor parte de esta heterogeneidad. De hecho el concepto en si de middleware tiene su origen en el desarrollo de sistemas distribuidos, donde las aplicaciones se ejecutan sobre una nube de sistemas y son por lo tanto clientes de servicios de intercambio de información confiable. Considerando

la relación entre el middleware y el nivel de acoplamiento de los sistemas, o el tipo de paralelismo entre los componentes de los sistemas distribuidos, puede observarse que el middleware está relacionado principalmente con los sistemas débilmente acoplados. El paralelismo masivo no requiere de middleware, aunque podría usarlo: en sistemas fuertemente acoplados, la comunicación entre los procesadores se hace a través de interfaces de bajo nivel (memoria compartida), intercambiando gran cantidad de datos a alta velocidad. En tales sistemas paralelos, la palabra clave para la relación entre procesadores es simplicidad y eficiencia y no el nivel de funcionalidad.

Por otro lado, la tarea de compartir información en un sistema distribuido implica generalmente el intercambio de datos más complejos y el uso de mecanismos de comunicación y sincronización más avanzados. Por lo tanto, hay una necesidad real para tener a disposición de las aplicaciones, interfaces de alto nivel para la implementación de la comunicación requerida entre procesos similar a aquella propuesta en las capas superiores del modelo de referencia de Interconexión de Sistemas Abiertos (modelo OSI).

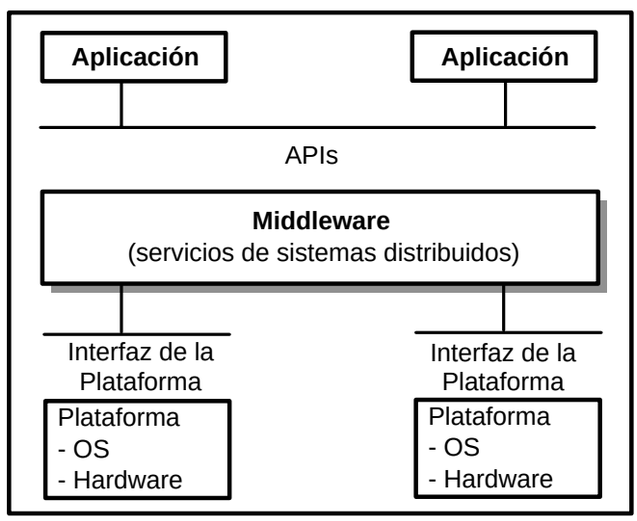


Figura 2.1-1: Concepto de Middleware.

Lo anterior nos conduce a la siguiente definición [3]:

Middleware es una capa de software que reside entre las aplicaciones de negocio y la capa de red de protocolos y plataformas heterogéneas. Desacopla las aplicaciones de negocio de cualquier dependencia de la capa formada por sistemas operativos heterogéneos, plataformas de hardware y protocolos de comunicación.

Middleware puede ser visto como un conjunto de funciones y servicios reutilizables y extensibles que son necesarios para que muchas aplicaciones funcionen bien dentro de un ambiente de red [4]. Esto incluye computación distribuida, bases de datos distribuidas, servicios de videos avanzados, comercio electrónico, etc..., los cuales manejan una gran cantidad de datos e información. Maneja las conversiones entre clientes y servidores, servidores y servidores, y, potencialmente, entre clientes y clientes.
Otra de las virtudes de cualquier Middleware es la de proteger al desarrollador de la dependencia de los protocolos de comunicación y plataformas de sistemas operativos y hardware. El desarrollador trabaja a un nivel más alto, ajeno a los detalles de bajo nivel.

Un middleware robusto y escalable constituye una infraestructura que posee la capacidad de lograr que los diversos componentes de computación de la empresa sean vistos desde un único punto de administración [5]. Este *middleware* debe tener la capacidad de ejecutarse en diferentes plataformas, crecer según las necesidades de la empresa(escalable) y permitir la completa integración entre los diferentes niveles de computación y las herramientas que sean utilizadas.

En esencia, middleware es un software de conexión, que simplifica el desarrollo de aplicaciones entre sistemas dispares en un ambiente de red cliente/servidor, es decir, puede ser considerado como una herramienta de integración de sistemas.
El middleware deberá proveer los niveles de seguridad que sean necesarios para garantizar altos estándares de integridad de la información y un alto nivel de seguridad para garantizar que la información está siendo utilizada por la persona adecuada en la tarea adecuada.

Dentro de las principales características con las que debe contar un middleware que intente apoyar la administración de la empresa se tienen [5]:

- Balancear las cargas de trabajo entre los elementos de computación disponibles.

- Manejo de mensajes, que le permite entrar en el modo conversacional de un esquema Cliente/Servidor y en general de cualquier forma de paso de mensajes entre procesos.
- Administración global, como una sola unidad computacional lógica.
- Manejo de consistencia entre diferentes manejadores de base de datos principalmente en los procesos de OLPT (Procesamiento de Transacciones en Línea).
- Administración de alta disponibilidad de la solución.

En resumen, podemos afirmar que un middleware puede lograr tres grandes objetivos en una empresa :

- Integración de Sistemas Heterogéneos.
- Interoperabilidad de estos sistemas.
- Portabilidad de estos sistemas.

Los beneficios directos que aporta el middleware son los siguientes:

- Mejora la productividad de programación.
- Baja el costo y tiempo de los proyectos.
- Construye ventajas competitivas.
- Apoyo a los sistemas abiertos y flexibles
- Permite el amplio uso de aplicaciones empresariales.
- Obtención de aplicaciones portables.
- Reducir el nivel de experiencia requerida para desarrollar y mantener aplicaciones distribuidas.

Las desventajas que puede presentar un middleware son:

1. Todos los nodos deben implementar productos de middleware de un mismo vendedor.
2. Las aplicaciones pueden usar sólo protocolos de transporte específicos de un mismo vendedor.
3. Las aplicaciones heredadas requieren de un desarrollo para nuevos estándares.

Por último, se debe mencionar que hay quienes confunden middleware con un conjunto de APIs. Sin embargo, el middleware es más amplio, abarca

más términos que se extienden entre la aplicación y la red de transporte. Al respecto, la figura 2.1-2 muestra la relación entre Middleware y un modelo de referencia OSI.

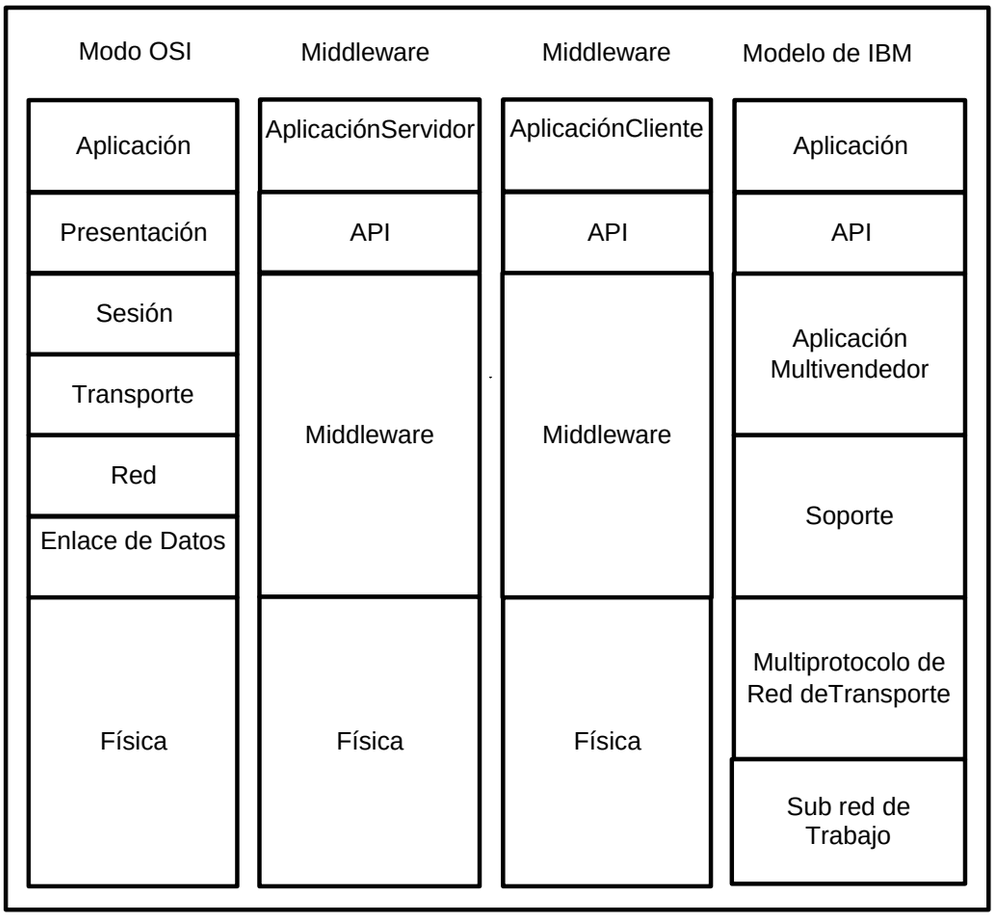


Figura 2.1-2. Concepto de Middleware en relación a un Modelo de Referencia OSI y a un Modelo de Comunicaciones de IBM.

2.2 MODELOS DE MIDDLEWARE

Es interesante entender las características generales de cada tipo de middleware para poder evaluar una tecnología específica. Hay dos tipos de modelos de middleware:

- Modelo Lógico.
- Modelo Físico.

El modelo de middleware lógico define cómo la información se mueve a través de una empresa desde un punto de vista conceptual. El modelo de middleware físico por su parte, muestra cómo la información realmente se mueve de acuerdo a la tecnología que utiliza.

Un análisis del modelo de middleware lógico requiere una discusión de configuraciones:

- Uno a uno
- Muchos a muchos

Así como también de los mecanismos de comunicación:

- Sincrónica.
- Asincrónica.

2.2.1 Middleware Lógico Punto a Punto

El middleware punto a punto es aquel que permite que una aplicación se enlace directamente a otra aplicación – la aplicación A se enlaza a la aplicación B usando un simple canal de comunicación. Cuando la aplicación A desea desconectarse simplemente envía un mensaje indicando la acción a través del canal de comunicación.

Las limitaciones que este tipo de middleware impone respecto de otros es su incapacidad para unir al mismo tiempo más de dos aplicaciones. También la falta de servicios para capas de procesamiento intermedios, que sería deseable en casos donde se necesiten lógica de negocio específico.

Hay varios ejemplos de middleware punto a punto, incluyendo productos MOM (tal como MQSeries) y RPCs (tal como DCE). El propósito de estos productos es proveer soluciones punto a punto, básicamente involucrando una aplicación origen y una aplicación destino. Aunque hoy en día es posible unir más de dos aplicaciones usando un middleware punto a punto tradicional, hacer esto no es buena idea. Se originan demasiadas complejidades cuando se involucran más de dos aplicaciones. Para poder unir más de dos aplicaciones bajo este esquema siempre será necesario un mecanismo de enlace punto a punto entre todas las aplicaciones.

2.2.2 Middleware Lógico Muchos a Muchos

Como su nombre lo dice, el middleware muchos a muchos enlaza varias aplicaciones con muchas otras. Esta es la tendencia en el mundo del middleware. Es además, el modelo de middleware lógico más potente, por cuanto provee flexibilidad y aplicabilidad a diversos dominios de problemas.

Hay varios ejemplos de middleware muchos a muchos, incluyendo broker de mensajes, middleware transaccional (servidores de aplicación y monitores TP) y, hasta objetos distribuidos. Básicamente, cualquier tipo de middleware que tenga la capacidad de trabajar con más de dos aplicaciones origen y destino al mismo tiempo, es capaz de soportar este modelo.

Así como la ventaja del modelo punto a punto es su simplicidad, la desventaja del modelo muchos a muchos es la complejidad de unir varios sistemas.

2.2.3 Mecanismos de Comunicación Sincrónica

Las comunicaciones sincrónicas ocurren cuando la comunicación entre un emisor y un receptor se han ejecutado de una manera coordinada. Esto exige que el emisor y el receptor para poder operar dependan del procesamiento de un requerimiento. Requiere de un emisor y de un receptor para coordinar su procesamiento interno en conjunto con las comunicaciones. Esta coordinación implica, que la comunicación sincrónica tiene un alto grado de acoplamiento. Las reglas para esta coordinación son dependientes del tipo de comunicación sincrónica usada.

La comunicación sincrónica es preferible en situaciones donde el emisor necesita el resultado del procesamiento desde el receptor o requiere de una notificación de recepción. Los sistemas interactivos requieren comunicaciones sincrónicas.

Hay tres tipos más comunes de comunicación sincrónica:

- Requerimiento/Respuesta
- Unidireccional
- Polling

La comunicación sincrónica se requiere para mandar aplicaciones que necesitan una respuesta desde una aplicación receptora para continuar con su procesamiento. Cada uno de los tipos mencionados anteriormente, manejan en forma distinta sus requerimientos. A continuación en forma breve explicaremos cada uno de ellos.

2.2.3.1 Requerimiento / Respuesta (Request / Reply)

Es el patrón básico de la comunicación sincrónica. En la comunicación requerimiento/respuesta, una aplicación envía un requerimiento a una segunda aplicación y se bloquea hasta que la segunda aplicación envía una respuesta. El bloqueo se produce debido a que necesitamos esperar la respuesta desde el receptor. La respuesta puede ser cualquier cosa, desde el reconocimiento de la recepción para el procesamiento completo con una respuesta. Después de la recepción de la respuesta el emisor continúa con su procesamiento. A continuación se muestra un ejemplo.

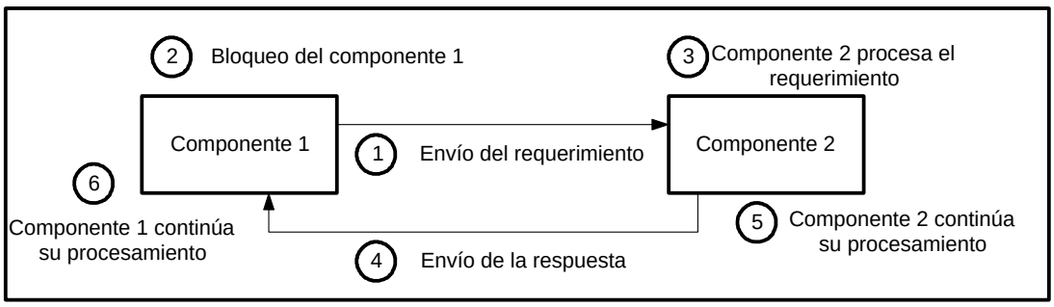


Figura 2.2-1. Comunicación Sincrónica en Requerimiento/Respuesta.

Este método es usado en situaciones en la cual la respuesta contiene información que es necesaria para que el emisor pueda continuar su procesamiento. Si el procesamiento del requerimiento en el receptor toma una cantidad de tiempo considerable, entonces el impacto del rendimiento puede ser sustancial y potencialmente inaceptable. Si el receptor tiene un problema y no es capaz de terminar el requerimiento, entonces el emisor no podrá continuar con todo el procesamiento.

2.2.3.2 Unidireccional

El patrón unidireccional es más simple que el requerimiento/respuesta, porque envía un requerimiento y se bloquea sólo hasta que recibe el reconocimiento de éste. La comunicación unidireccional es una forma de comunicación sincrónica donde el emisor hace un requerimiento a un receptor y espera por una respuesta de conocimiento de recepción del requerimiento. La figura 2.2-2 muestra un ejemplo.

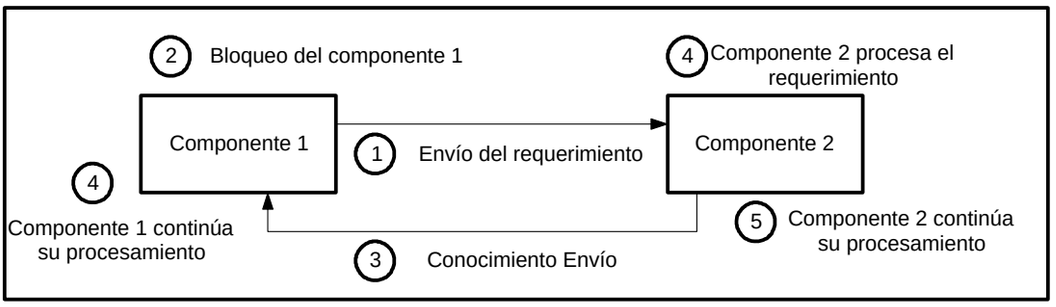


Figura 2.2-2. Comunicación Sincrónica Unidireccional.

Este método es usado en situaciones donde el emisor debe sincronizar su procesamiento con el receptor. El emisor no puede continuar su procesamiento hasta que el receptor ha recibido el requerimiento. La primera desventaja de este método es que no hay otra información que el envío del conocimiento al emisor por el receptor. La segunda desventaja son las implicaciones sobre el rendimiento que tiene el bloqueo del emisor. La recepción del conocimiento debería ser mucho más rápida, sin embargo hay que esperar por el procesamiento completo y entonces recibir una respuesta.

2.2.3.3 Polling

Polling es un mecanismo de comunicación sincrónico que permite al emisor continuar con el procesamiento de una manera limitada mientras espera por una respuesta del receptor. En este patrón de comunicación, el emisor envía el requerimiento, pero en lugar de bloquearse por la respuesta continúa con su procesamiento. Debe parar y revisar por la respuesta periódicamente. Una vez que ha recibido la respuesta puede continuar procesando sin seguir preguntando.

Por supuesto, este caso es útil sólo cuando el emisor tiene algo útil que hacer mientras espera por la respuesta. La figura 2.2-3 muestra un ejemplo.

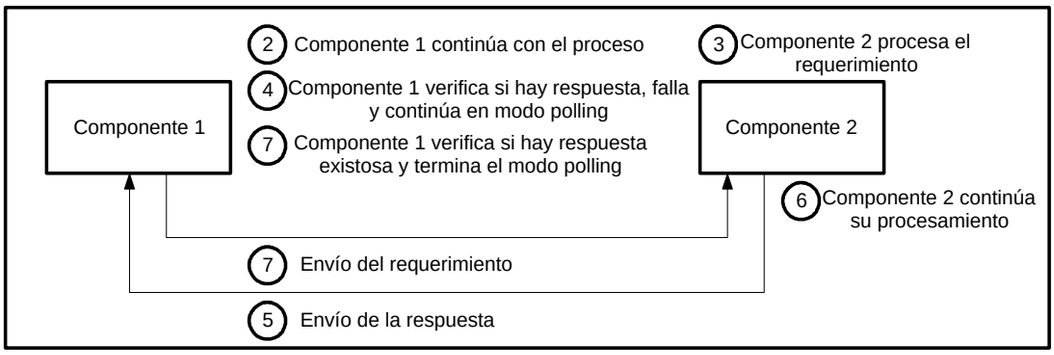


Figura 2.2-3. Comunicación Sincrónica de Polling.

2.2.4 Mecanismos de Comunicación Asincrónica

La comunicación asíncrona ocurre cuando la comunicación entre un emisor y un receptor es realizada de manera tal que permite a cada uno de ellos operar independientemente del otro. El receptor del requerimiento no posee la obligación de manejar las comunicaciones o responder al emisor. El emisor continúa operando una vez que el requerimiento se ha enviado independiente de cómo el receptor maneja la comunicación.

Provee un grado más bajo de acoplamiento que la comunicación sincrónica. No es responsabilidad del emisor si el mensaje fue recibido, cómo es procesado, o el resultado del receptor como parte de la comunicación. Este modelo se utiliza cuando la comunicación de información no requiere la coordinación de actividades o respuestas.

La comunicación asincrónica es útil cuando el propósito de la comunicación es la de transferir información. Además, puede operar en ambientes no confiables donde las redes y servidores pueden no estar siempre disponibles. Ejemplos pueden incluir sistemas donde una actualización se envía desde una aplicación a todas las otras que tienen copia del mismo dato, o cuando un cambio ha ocurrido en una base de datos. Otro ejemplo lo constituye la ocurrencia de un evento que genera la notificación de otras aplicaciones.

A continuación se analizarán los tres tipos más comunes de comunicación asincrónica:

- Intercambio de Mensajes.
- Publicar / Subscribir.
- Broadcast.

2.2.4.1 Intercambio de Mensajes

El intercambio de mensajes es una forma de comunicación asincrónica donde un requerimiento se envía desde un emisor a un receptor. Cuando el emisor ha hecho el requerimiento esencialmente se olvida que ha sido enviado y continúa su procesamiento. El requerimiento es entregado al receptor para su procesamiento.

De acuerdo a la figura 2.2-4, el componente 1 crea y envía un mensaje y continua su procesamiento:

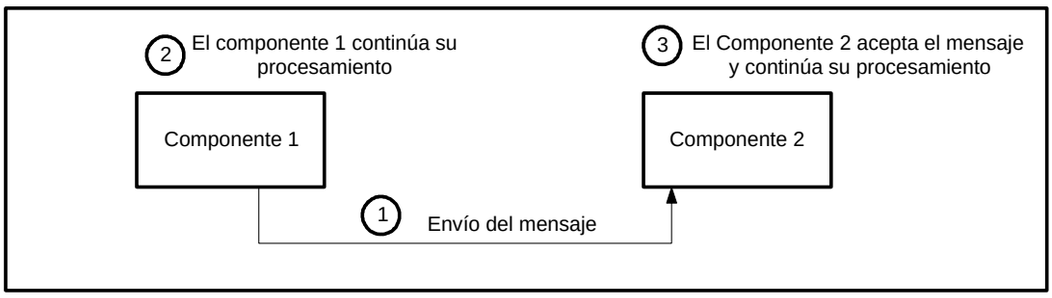


Figura 2.2-4. Intercambio de mensajes asincrónicos.

Esta es la forma más simple de comunicación asincrónica. Para que sea efectivo este mecanismo de comunicación deberá ser implementado sobre una red confiable o con un servicio que provea una garantía de la entrega. Este servicio debe ser capaz de continuar tratando de enviar el requerimiento a través de la red al receptor hasta que sea capaz de completar la comunicación.

Este método se usa en situaciones donde la información necesita ser transmitida sin que medie una respuesta.

2.2.4.2 Publicar / Subscribir (Publish / Subscribe)

Publicar y subscribir es una forma de comunicación asincrónica donde un requerimiento es enviado por el emisor y, donde el receptor se determina basándose en una declaración de interés (del receptor) en un requerimiento (subscripción) previamente definido.

Publicar / subscribir permite determinar la dirección de un requerimiento basado sobre el interés del receptor. El receptor se subscribe a requerimientos a través de una declaración de interés que describe atributos

de un requerimiento que desea recibir. Este mecanismo se muestra en la figura 2.2-5.

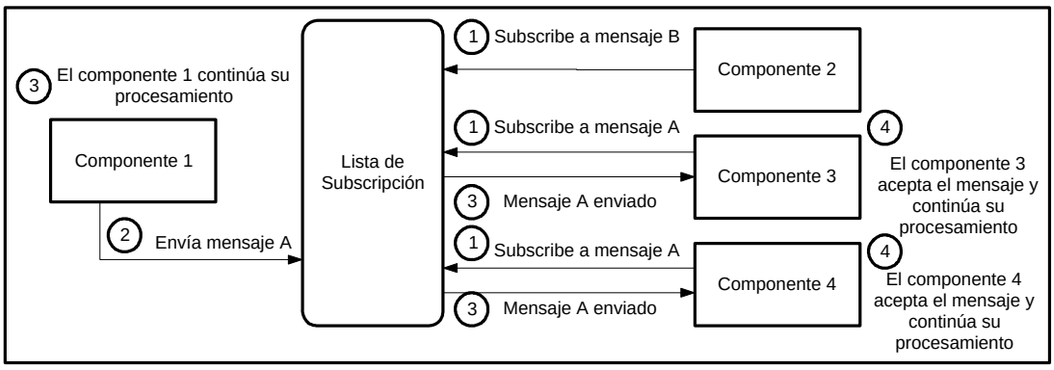


Figura 2.2-5. Mecanismo de comunicación asincrónica Publicar/Subscribir.

Publicar / subscribir permite a cada aplicación en el sistema decidir a cerca de qué eventos desea ser notificado. Esto se realiza a través de la definición de información, estructura de datos, o tipos de requerimientos en los cuales está interesado recibir. Este método se usa en situaciones donde no se requiere una respuesta y el receptor es determinado por el contenido del requerimiento.

Este patrón es útil en un sistema de integración de procesos de múltiples pasos donde el requerimiento es realmente la notificación de que un evento ha ocurrido – por ejemplo, la notificación de la orden de proceso cuando una orden de un producto ha sido enviada o pagada.

2.2.4.3 Broadcast

Este es un mecanismo de comunicación asincrónica, en el cual el requerimiento se envía a todos los participantes, que denominaremos receptores de una red. Cada participante determina si el requerimiento es de su interés examinando su contenido.

Como lo muestra la figura 2.2-6, el mensaje se envía a cada aplicación en el sistema y es ésta la que determina si le conviene el mensaje o no. En caso que el mensaje le interese, la aplicación procederá con el procesamiento del requerimiento de acuerdo a la lógica programada en el receptor, en caso contrario, será ignorado.

Este mecanismo de mensaje necesita ser utilizado con mucho cuidado porque puede convertirse en un cuello de botella debido a que cada posible

receptor debe analizar el mensaje para determinar si debe proceder con su procesamiento o no.

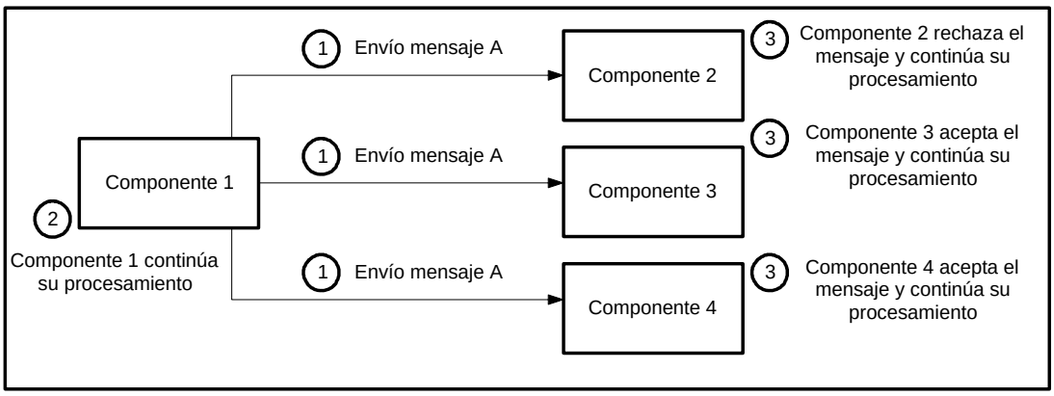


Figura 2.2-6. Modelo broadcast de comunicación asincrónica.

2.3 CLASIFICACIÓN DE MIDDLEWARE

2.3.1 Middleware Orientado a Mensajes (MOM)

2.3.1.1 Conceptos

Los *middleware* MOM manejan eventos asincrónicos, sin bloqueo y métodos de comunicación basados en mensajes que garantizan la entrega de estos [6].

El término “mensajería” normalmente se asocia a sistemas de correo electrónico. Sin embargo, los servidores MOM difieren radicalmente de este, porque ellos son de alta velocidad, generalmente sin conexión y están usualmente distribuidos para la ejecución de aplicaciones concurrentemente con un envío no bloqueado. El *middleware* de encolamiento (MQM) es un tipo de MOM que combina el movimiento de mensajes de alta velocidad y servicios de almacenamiento de mensajes.

Los productos MOM podrían ser medidos en términos de: funcionalidad, rango de apoyo a plataformas y redes, costo de propiedad, apoyo para las herramientas de programación, facilidad de instalación, actualización y funcionamiento.

La mensajería tiene un método asincrónico de paso de información entre programas, pero también existen productos MOM que soportan el estilo sincrónico de comunicación.

Los mensajes pueden ser persistentes y no persistentes, lo cual va a depender de las características de las colas donde se almacenen. Se

entiende por persistente a aquellos que están escritos en una cola de almacenamiento no volátil, desde donde ellos pueden ser restaurados después que el sistema se haya restablecido. Los mensajes no persistentes son almacenados en colas que residen en memoria.

El encolamiento de mensaje tiene un método indirecto de paso de información a través de colas de mensajes, o sea, las herramientas MOM utilizan colas administradas por servidores o servicios que manejan la entrada y salida de las colas, estos no llegan en forma directa. Los mensajes son almacenados en colas hasta que el recipiente está listo para leerse.

Cabe mencionar que los diferentes vendedores tienen diferentes APIs y no son fáciles de interoperar entre uno y otro, y cada uno de ellos utiliza sus propias propiedades de ubicación de los servicios (directorio, nombre y seguridad).

2.3.1.2 Colas de Mensajes

Las Colas de Mensajes constituyen un modelo de comunicación indirecto dentro del MOM que permite a los programas comunicarse a través de colas de mensajes [3]. Las colas de mensajes siempre implican un modelo orientado a la desconexión. Por lo tanto, la disponibilidad de uno de los programas no es obligatoria.

Como se muestra en la figura 2.3-1, los mensajes son puestos en colas(las cuales pueden estar en memoria o estar basadas en disco) para su entrega inmediata o posterior. Esto permite la ejecución independiente de los programas, a diferentes velocidades y sin una conexión lógica entre ellos.

A pesar de las diferentes implementaciones encontradas hoy en día en los productos de colas de mensajes, la mayoría de ellos incorpora la siguiente funcionalidad:

- Generalmente, los productos de colas de mensajes exponen APIs que los programadores de aplicación usan para facilitar el intercambio de mensajes. Normalmente se habla de enviar y recibir mensajes a y desde las colas.
- El componente implícito de un típico sistema de colas de mensajes es un *Administrador de Colas*. Maneja las colas locales y garantiza que los mensajes serán transferidos a su destino final, ya sea sobre la misma máquina o sobre una distinta en la red. Otras funciones de control que

realiza el administrador de colas incluyen diferentes niveles de confirmación de recepción de mensajes, priorización y balanceo de carga.

- El administrador de colas colabora con otros administradores de colas que podrían estar sobre nodos distintos para controlar el camino a través de la red(Encontrar rutas alternativas cuando el camino no está disponible).
- Las colas de mensajes implican el soporte de diferentes niveles de Calidad de Servicio. Esas calidades de servicio pueden ser:
  1. Entrega de mensajes confiable, durante el intercambio de mensajes no hay pérdidas de paquetes.
  2. Entrega de mensajes garantizada, los mensajes son entregados al nodo destino en forma inmediata(cuando no hay latencia, disponibilidad de la red) o, eventualmente, con posterioridad(con latencia, red no disponible). En el último caso, el middleware garantiza que los mensajes son entregados tan pronto como la red esté disponible dentro de un período de tiempo especificado.
  3. Asegura la entrega de mensajes no duplicados, si los mensajes son entregados, ellos son entregados sólo una vez.

- Las colas de mensajes pueden ser persistentes o no persistentes. En el último caso los mensajes se pierden en el caso que el Administrador de Colas falle. En el otro caso, los mensajes son recuperados una vez que se reinicia el Administrador de Colas. Resulta natural pensar que en aplicaciones donde la entrega de los mensajes es crítica, se utilizará un esquema basado en colas persistentes.
- Algunos productos de colas de mensajes soportan triggers, donde un programa de aplicación es activado sólo cuando un mensaje de requerimiento o un mensaje de respuesta ha llegado a la cola local. Esta característica permite a las aplicaciones estar activas sólo cuando hay trabajo que hacer, lo cual evita el consumo de recursos innecesarios.

Una característica avanzada que está disponible sobre productos competitivos es la noción de mensaje transaccional. La semántica transaccional puede ser aplicada a los actos de encolar y desencolar a través de múltiple colas, bajo el control del Administrador de Colas. Esto

permite una división de una transacción sincrónica que abarca múltiples nodos de una red en una cantidad de transacciones más pequeñas. Esas transacciones más pequeñas operarán asincrónicamente y estarán dirigidas por un mensaje asincrónico. Esto presenta una alternativa al uso de transacciones sincrónicas con el protocolo two-phase commit. Sin embargo, debe mencionarse que la mayoría de los productos de colas de mensajes sólo operan con transacciones sobre sus colas distribuidas y no sobre otra clase de recursos, como podría ser una base de datos.

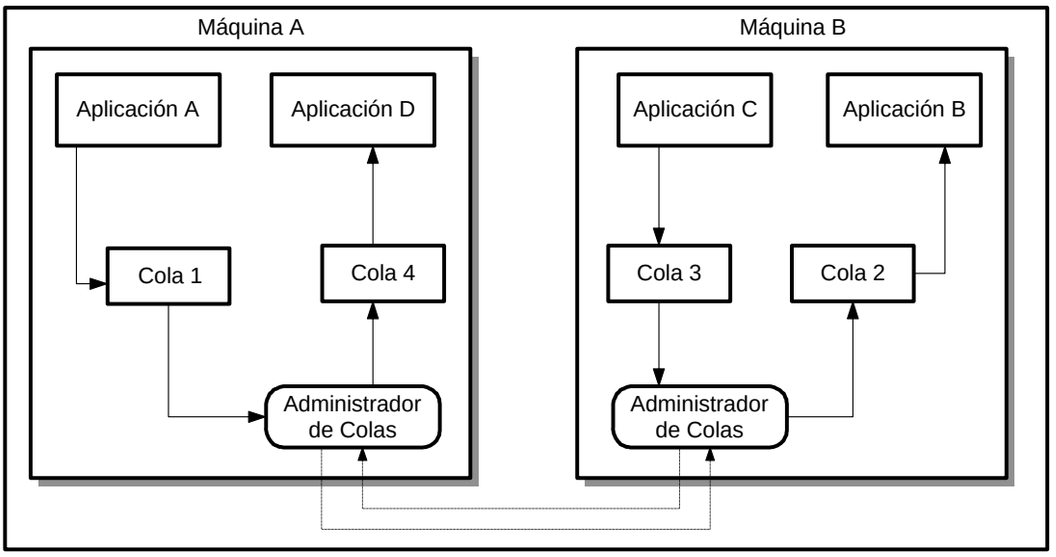


Figura 2.3-1. Modelo de Middleware Orientado a Mensajes(MOM).

2.3.2 Llamada a Procedimientos Remotos(RPC)

2.3.2.1 Conceptos

La idea de Llamadas a Procedimientos Remotos(RPC) es muy simple. Está basada en la observación de que las llamadas a procedimientos constituyen un mecanismo bien conocido y bien entendido para la transferencia de control y datos dentro de un programa ejecutándose en un computador. Por lo tanto, se ha propuesto extender [7] este mismo mecanismo para proveer transferencia y control de datos a través de una red de comunicaciones. Cuando se invoca una llamada a procedimiento, el ambiente que hace la llamada se suspende, los parámetros son pasados a través de la red al ambiente donde el procedimiento se ejecuta. Una vez ejecutado el procedimiento, el resultado de dicha operación es devuelto al ambiente que

generó el llamado, continuando su ejecución como si se tratara de una llamada local.
RPC es una infraestructura basada en el modelo cliente/servidor que aumenta la interoperabilidad, portabilidad y flexibilidad de una aplicación al permitir que ésta pueda distribuirse sobre diversas plataformas heterogéneas. Reduce la complejidad del desarrollo de aplicaciones que abarcan diferentes sistemas operativos y protocolos de red, al aislar al programador de los detalles de estos entornos

Hay que indicar que en la actualidad hay tres especificaciones principales de RPC [8] en el mercado:

- RPC ONC, algunas veces denominada como RPC de Sun, fue una de las primeras implementaciones comerciales de RPC. Hay básicamente dos implementaciones de RPC ONC, la implementación original y una implementación independiente del transporte. La implementación original normalmente está disponible sobre la mayoría de los sistemas como parte del software de red del sistema. Por otro lado, la implementación más reciente de RPC ONC es TI RPC(Transport Independent RPC), la cual está disponible principalmente como parte del sistema operativo Solaris. La mayor diferencia entre estas dos implementaciones es que TI RPC puede utilizar diferentes protocolos a nivel de la capa de transporte.

- RPC DCE(*Distributed Computing Environment*), de la OSF(*Open Software Foundation*).

- RPC ISO de la Organización Internacional para la Estandarización.

2.3.2.2 Modelo RPC

El modelo RPC describe la manera en que procesos cooperando sobre diferentes nodos de una red pueden comunicarse y coordinar actividades. El paradigma RPC está basado en el concepto de una llamada a procedimiento en un lenguaje de programación. La semántica de RPC es casi idéntica a la semántica de una llamada a procedimiento tradicional. La mayor diferencia es que mientras una llamada a procedimiento normal tiene

lugar entre procedimientos de un solo proceso en el mismo espacio de memoria sobre un solo sistema, RPC tiene lugar entre un proceso del cliente en un sistema y un proceso del servidor en otro sistema y dónde el sistema del cliente y el sistema del servidor se conectan a través de una red.

La figura 2.3-2 ilustra la operación básica de RPC. Una aplicación cliente genera una llamada a procedimiento normal a un stub(*código ficticio*) del cliente. El stub del cliente recibe argumentos de la llamada al procedimiento y devuelve los argumentos resultantes de la llamada. Los argumentos en este contexto pueden ser de entrada, salida o de entrada / salida.

El stub del cliente convierte los argumentos de entrada desde una representación de datos local a una representación de datos común, crea un mensaje conteniendo los argumentos de entrada en su representación de datos común. El ambiente de ejecución del Cliente transmite el mensaje con los argumentos de entrada al ambiente de ejecución del servidor, el cual es normalmente una librería de objetos que da soporte a la funcionalidad del stub del servidor. El ambiente de ejecución del Servidor emite una llamada al stub del servidor el cual toma los argumentos de entrada del mensaje, los convierte desde la representación de datos común a la representación de datos local del servidor para luego hacer la llamada a la aplicación del servidor que hace el procesamiento del requerimiento.

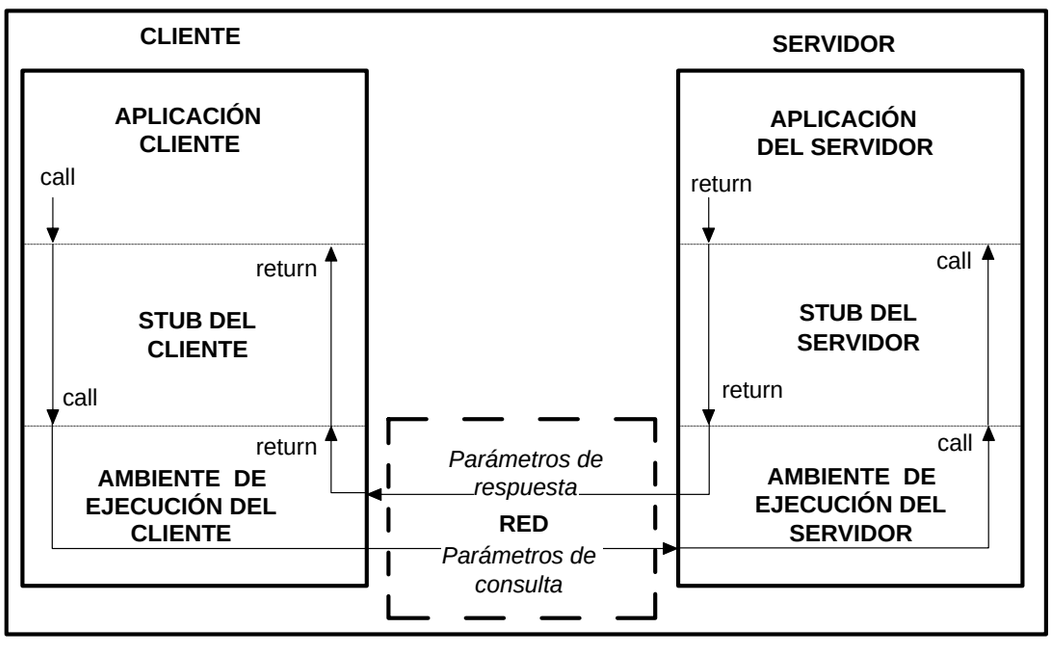


Figura 2.3-2: Modelo de Middleware RPC.

2.3.2.3 Implementación RPC

Una implementación de un modelo RPC consiste normalmente de al menos tres elementos: un compilador de lenguaje, un ambiente de ejecución para el cliente y un ambiente de ejecución para el servidor. El compilador del lenguaje produce stubs tanto para el cliente como para el servidor a partir de un programa escrito en un lenguaje RPC, el cual normalmente es un lenguaje no procedural que posee la capacidad de declaración de procedimientos remotos y sus respectivos parámetros. En conjunto con las aplicaciones cliente y servidor, los stub del cliente y servidor son compilados por un compilador de lenguaje procedural, tal como C, produciendo archivos de objetos los cuales son enlazados a las librerías del ambiente de ejecución del cliente y del servidor. Este proceso produce un ejecutable para el cliente y un ejecutable para el servidor.

Las librerías del ambiente de ejecución tanto del cliente como del servidor son llamadas por los stubs del cliente y servidor respectivamente. Esos objetos de las librerías del ambiente de ejecución contienen las rutinas para realizar la conversión entre la representación de datos local y la representación de datos común, para la creación de los formatos de mensajes que viajarán por la red y, para la transmisión de esos mensajes entre el cliente y el servidor de acuerdo a los protocolos específicos del usuario.

Con tal implementación, el desarrollador de una aplicación RPC requiere escribir lo siguiente:

- El programa en lenguaje RPC para el compilador del lenguaje RPC.
- La aplicación cliente la cual llama al stub del cliente.
- La aplicación servidora la cual es invocada por el stub del servidor.

2.3.2.4 Variaciones en el Modelo Requerimiento/Respuesta

Una implementación RPC puede soportar diferentes variaciones sobre el modelo requerimiento/respuesta. Estas variaciones pueden incluir un RPC tipo broadcast y un RPC en el cual no se requiere respuesta ante un requerimiento.

- Broadcast, una variación de RPC tipo broadcast permite a las aplicaciones hacer llamadas a más de un servidor con una sola invocación RPC.

- Sin respuesta, una variación RPC sin respuesta permite hacer una llamada RPC en la cual no se recibe respuesta y por tanto, permite seguir con la ejecución del cliente. Un ejemplo puede ser un proceso que esté monitoreando alguna actividad. El proceso monitor hace llamadas RPC a un servidor el cual mantiene un log de la actividad que está siendo monitoreada. Por lo que el cliente no necesita una respuesta de la actividad que está siendo registrada.

2.3.2.5 Portabilidad e Interoperabilidad de Aplicaciones RPC

Entre otras cosas, los estándares de procesamiento de información proveen portabilidad e interoperabilidad. Por lo anterior, es conveniente hacerse la pregunta respecto a qué elementos del modelo y/o implementación RPC son candidatos para la estandarización. Basados en el modelo de la figura 2.3-2, las especificaciones estándares podrían ser desarrolladas para un lenguaje RPC, un protocolo RPC, o las librerías del ambiente de ejecución del cliente y servidor. Lo anterior no impide que otros aspectos de RPC también estén sujetos a estandarización. Se debe indicar que entre las distintas versiones de RPC existentes(RPC de ONC, RPC de DCE y RPC de ISO) no hay compatibilidad.

El particular, el RPC de ISO especifica un lenguaje RPC y un protocolo RPC. La especificación tanto del lenguaje RPC como del protocolo RPC evita que un programa en lenguaje RPC pueda ser portado a otro sistema, por cuanto los dos sistemas no pueden interoperar mientras los ambientes de ejecución(runtime) del cliente y del servidor no usen el mismo protocolo RPC.

En cuanto a la portabilidad de aplicaciones RPC, esta se lleva a cabo a nivel de código fuente por medio del lenguaje RPC. Hay que notar, que cuando una especificación RPC define sólo un lenguaje y un protocolo, no necesariamente los stubs del cliente y del servidor(la salida del compilador del lenguaje RPC) son portables. Los stubs del cliente y del servidor llaman

a rutinas en las librerías de runtime del cliente y del servidor las cuales son probablemente específicas al sistema que contiene la implementación de la especificación RPC. Sobre otro sistema, el compilador de lenguaje RPC genera los stubs del cliente y servidor apropiados para las librerías del ambiente de ejecución de ese sistema. Por lo tanto, los stubs del cliente y del servidor no son portables por sí mismos a otro sistema.

En consecuencia, para que una aplicación RPC sea portable cuando una especificación RPC define sólo un lenguaje y un protocolo, es necesario que los stubs del cliente y del servidor estén completos. En nuestro contexto, los términos *stub del cliente completo* y *stub del servidor completo* se refiere a que los stub generados por el compilador de lenguaje RPC no deben ser modificados por el desarrollador de la aplicación para ser integrados con las aplicaciones clientes y servidoras. Por lo anterior, si un desarrollador de aplicaciones debe modificar la salida del compilador del lenguaje RPC, entonces la portabilidad de la aplicación se verá disminuida.

Por último, la interoperabilidad se logra a través del protocolo RPC. La aplicación RPC tiene acceso al protocolo RPC a través de las librerías del ambiente de ejecución del cliente y del servidor. Debido a que el protocolo RPC es parte de la especificación estándar RPC, una aplicación cliente sobre un sistema interopera con un servidor sobre cualquier otro sistema.

2.3.3 Arquitectura de Objetos Distribuidos

2.3.3.1 Conceptos

CORBA (Common Object Request Broker Architecture), “Arquitectura de bus común de gestión de requerimientos de objetos” es un marco de trabajo estándar de Objetos Distribuidos creado por el consorcio OMG(Object Management Group), un consorcio de más de 760 empresas que se creó en 1989 con fundadores como Hewlett-packard y Sun Microsystems. Esta es una organización sin ánimo de lucro cuyos fines son promover la Orientación a Objetos en la ingeniería del software y el establecimiento de una plataforma de arquitectura común para el desarrollo de programas basados en Objetos Distribuidos.

El primer paso en el camino del OMG fue la definición de la OMA (Object Management Architecture). Esta arquitectura representa el modelo fundamental en el que el resto de las tecnologías de la OMG se basan. La OMA establece un marco en el que se incluye:

- Un Modelo de Objetos base (Core Object Model), que define los elementos básicos de la programación orientada a objetos (clases, objetos, implementación, interfaces, invocación, cliente, servidor, etc.)

- Un Modelo de Referencia, que establece el marco de la arquitectura. Este identifica cinco elementos principales:

  - Un Bus común de gestión de requerimientos y respuestas entre objetos en el que pueden ser integrados componentes de forma estándar. Este bus se ha denominado ORB (Object Request Broker), y se documenta en los estándares de CORBA.

  - Servicios de Objetos, aquí se identifican un conjunto de servicios disponibles para los objetos. Estos se especifican en los documentos CORBA Services. Entre los servicios, se incluyen el servicio de nombre, eventos, ciclo de vida, transacciones, etc.

  - Servicios Comunes, se documentan en CORBA Facilities y, especifican servicios comunes a todos los objetos, como documentos compuestos, servicios de agentes móviles, manejo de sistemas, etc.

  - Interfaces de Dominio, son marcos específicos para dominios de desarrollo, como por ejemplo, programas médicos, control de tráfico aéreo, etc. Todos ellos, al igual que los anteriores, especificados como interfaces.

  - Interfaces de Aplicación, estas son las interfaces que constituyen las aplicaciones desarrolladas en forma específica.

CORBA es un refinamiento del modelo OMA, es decir, extiende de manera específica las definiciones y conceptos que lo componen. Establece por ejemplo, la diferencia entre Implementación de Objetos y Referencia de Objetos, la semántica de las interfaces, la semántica de las llamadas a métodos, excepciones, etc. Además, como parte importante, define el lenguaje de especificación de interfaces: IDL (Interface Definition Languaje) es un lenguaje independiente del lenguaje de programación utilizada.

2.3.3.2 Caracteristicas de CORBA

A continuación se describirán algunas de las principales características de CORBA:

- Interfaces definidas utilizando IDL. La separación entre interfaz e implemetación en CORBA está implícita y se consigue definiendo todos los componentes de la aplicación e incluso los servicios genéricos disponibles utilizando un lenguaje de descripción independiente de la implementación: el IDL.

- Sistema de Meta-información. Gracias al repositorio de interfaces, un objeto CORBA puede acceder a toda la meta-información sobre las interfaces que definen los demás objetos que están presentes en el sistema. Esto permite a los objetos localizar servicios genéricos o comunicarse y conocer otros objetos nuevos que se van integrando en el sistema dinámicamente. Esto simplifica el proceso de desarrollo y modificación de las aplicaciones y convierte a todo el sistema en un gran repositorio para herramientas de programación.

- Generación de requerimientos en forma dinámica. Al poseer meta-información, los clientes tienen la capacidad de ejecutar métodos en forma dinámica sobre objetos nuevos o ya existentes.

- Independencia del lenguaje de programación. CORBA ofrece mecanismos que permiten que los métodos de los objetos definidos en IDL sean implementados en cualquier lenguaje de programación. Los clientes de estos objetos no poseen la capacidad de discernir en que

lenguaje fueron implementados los objetos que les proveen servicios. Esta independencia no sólo ayuda a que cada parte del sistema sea implementada en el lenguaje adecuado para su funcionalidad, sino que permite la integración de sistemas ya existentes (Sistemas heredados) a esta arquitectura de objetos distribuidos.

- Transparencia de localización y activación del servidor. El núcleo de CORBA, el ORB ofrece a los objetos un mecanismo por el que pueden realizar invocaciones sobre objetos remotos de forma que estas aparezcan ante el sistema como locales. El ORB se encarga de alcanzar los objetos del servidor reales y enrutar el requerimiento en beneficio del usuario. Esto permite desacoplar de la aplicación todo el manejo de la comunicación, dando la visión al programador de que es un solo sistema, independientemente si se trata de varias máquinas conectadas por redes heterogéneas.

- Generación automática de Stub y Skeleton. Los sistemas distribuidos requieren un alto grado de programación de bajo nivel para manejar el inicio, flujo y finalización de la comunicación, codificar y decodificar argumentos en el formato que son transmitidos, etc.; estos son los stubs del cliente y skeletons del servidor. A través de los compiladores de IDL, el código que analiza todas estas funciones se generan automáticamente. Un cliente de un objeto sólo necesita su IDL para construir de forma automática el código que le permitirá realizar invocaciones remotas de forma transparente.

- Reutilización de CORBA Services y CORBA Facilities. Los objetos que forman una aplicación distribuida normalmente requieren una serie de servicios adicionales. Estos servicios, en CORBA forman parte de lo que se conoce como CORBA Services y CORBA Facilities. Los servicios disponibles incluyen un servicio de localización de objetos por nombre, es decir, obtener una referencia del objeto conociendo su nombre único; servicios de localización de objetos por tipo, es decir, obtención de referencias de objetos en base a necesidades particulares, de manera similar a una búsqueda en las páginas amarillas; servicios de eventos,

en el que los objetos se pueden registrar para ser notificados de eventos de interés; servicios de transacciones, de seguridad, etc. Estos servicios siempre están disponibles para ser reutilizados por las aplicaciones una y otra vez, liberando así a los desarrolladores de la carga de tener que implementarlos.

- Independencia del fabricante a través de la interoperabilidad de los ORBs y la portabilidad del código. CORBA define un protocolo estándar a través del que varios ORBs de distintos fabricantes se pueden comunicar de forma estándar (GIOP, General Inter-ORB Protocol). Esto significa que cualquier ORB que sea compatible con CORBA puede ser accedido desde cualquier otro. El desarrollador puede utilizar el ORB del fabricante que dé mejores prestaciones de una plataforma o lenguaje específico, sabiendo que, sus objetos van a presentar una interfaz uniforme y compatible.

En definitiva, CORBA nos permite programar para conseguir funcionalidad, independientemente de en qué lenguaje, estación de trabajo o plataforma hardware esté implementada: siempre está “lista para usar”.

2.3.3.3 Introducción a CORBA

CORBA es una arquitectura estándar para el desarrollo de aplicaciones distribuidas basadas en Objetos. Permite que las clases que forman parte de las aplicaciones puedan ser implementados en distintos lenguajes, se ejecutan en distintas plataformas hardware + Sistema Operativo o estén dispersas por una red heterogénea.

Para conseguir esto, CORBA se centra en tres ideas claves:

- La separación entre interfaz e implementación. A través del IDL se especifican todos los componentes CORBA. IDL es un lenguaje puramente declarativo con una sintaxis muy parecida a la de C++, pero sin estructuras programáticas. Es independiente del lenguaje utilizado en la implementación, existiendo enlaces (bindings o mappings) para

diversos lenguajes de programación (C, C++, Java, Ada, Smalltalk, COBOL, etc.). Permite especificar las clases de las que un componente hereda, la signatura de operaciones, los atributos, las excepciones que lanza, y la signatura de métodos (incluyendo argumentos de entrada, de salida y valores de retorno y sus tipos de datos), etc.

- La independencia de localización. El núcleo y componente más importante de cualquier implementación CORBA es el ORB. Este se encarga de hacer transparente la localización de los objetos, enrutando los requerimientos de manera que un objeto pueda comunicarse con otros independientemente de si ambos objetos se ejecutan en la misma máquina o en otra a través de redes heterogéneas.

- La independencia del fabricante y la integración de sistemas a través de la interoperatividad. CORBA, se ha definido un estándar para que ORBs de distintos fabricantes puedan integrarse en organizaciones heterogéneas y escalables de Objetos Distribuidos. El protocolo GIOP (General Inter-ORG Protocol), y su específico para Internet, IIOP, permiten a ORBs de distintos fabricantes comunicarse de una manera estándar. Esto, de cara al programador ofrece dos beneficios: 1) la independencia del vendedor; y 2) una invocación de métodos independiente de si ambos objetos (cliente y servidor) están en el mismo o en distintos ORBs.

El ORB (Object Request Broker)

El ORB es el responsable de permitir a los objetos realizar de manera transparente las invocaciones y recibir respuestas de otros objetos en un ambiente distribuido, independiente si dichos objetos están en la misma máquina o en una distinta y si se trata de redes homogéneas o heterogéneas.
En el ambiente CORBA, para que un objeto cliente pueda invocar operaciones en un objeto del servidor, el objeto cliente debe obtener una referencia al objeto servidor. El proceso de invocación se divide en dos pasos:

- Obtención de la referencia al objeto remoto.
- Invocación de la operación que se necesita.

Una vez que el objeto ha obtenido una referencia a un objeto del servidor, puede realizar la llamada a los métodos y acceder a los atributos de este objeto, los que están definidos a su vez a través del IDL de la clase a la que pertenece.

La figura 2.3-3 muestra la estructura del ORB de CORBA. Esta nos permite localizar todos los componentes y establecer la serie de servicios que están disponibles para los objetos CORBA.

Como se ha manifestado, todos los servicios disponibles para los objetos CORBA están definidos utilizando IDL. En la figura 2.3-3 se pueden observar las interfaces que el ORB ofrece a todos los objetos CORBA. Como se ve, el propio ORB ofrece un conjunto de servicios comunes(bajo el nombre de *ORB Interface*) que pueden ser utilizados tanto por objetos clientes como por implementaciones. Entre las operaciones ofrecidas por la interfaz del ORB se encuentra la posibilidad de convertir referencias en cadenas de caracteres y viceversa para poder comunicar de manera simplificada referencias a objetos, obtener la clase de un objeto, etc.

Hay que mencionar que tanto los clientes como los servidores necesitan adaptadores que transformen, el lado del cliente, una invocación local a un requerimiento al ORB y, en la parte del servidor, una invocación por parte del ORB en una invocación en el objeto que implementa el método que se está llamando. En la parte cliente, estos trozos de código se denominan *Stubs* y en el lado del servidor se llaman Skeletons. CORBA también provee interfaces para construir invocaciones en forma dinámica. Tanto en el cliente(*Dynamic Invocation Interface*) como en el servidor quien implementa el objeto(*Dynamic Skeleton Interface*) se provee una interfaz para que los objetos puedan invocar métodos para las que no poseen el *stub*, especificando el objeto, el nombre de la operación o método que se invocará sobre él y los parámetros de la llamada. Estos servicios dinámicos se apoyan en la meta-información dada por el *Repositorio de Interfaz*. Finalmente, el adaptador de objetos(Object Adapter) se apoya en Repositorio de Implementación para controlar el registro de servidores,

activación y desactivación de implementaciones, manejo de instancias en base a diversos criterios, etc.
A continuación se verá con más detalle estos componentes, lo cual permite tener una visión global de los distintos servicios que ofrece CORBA.

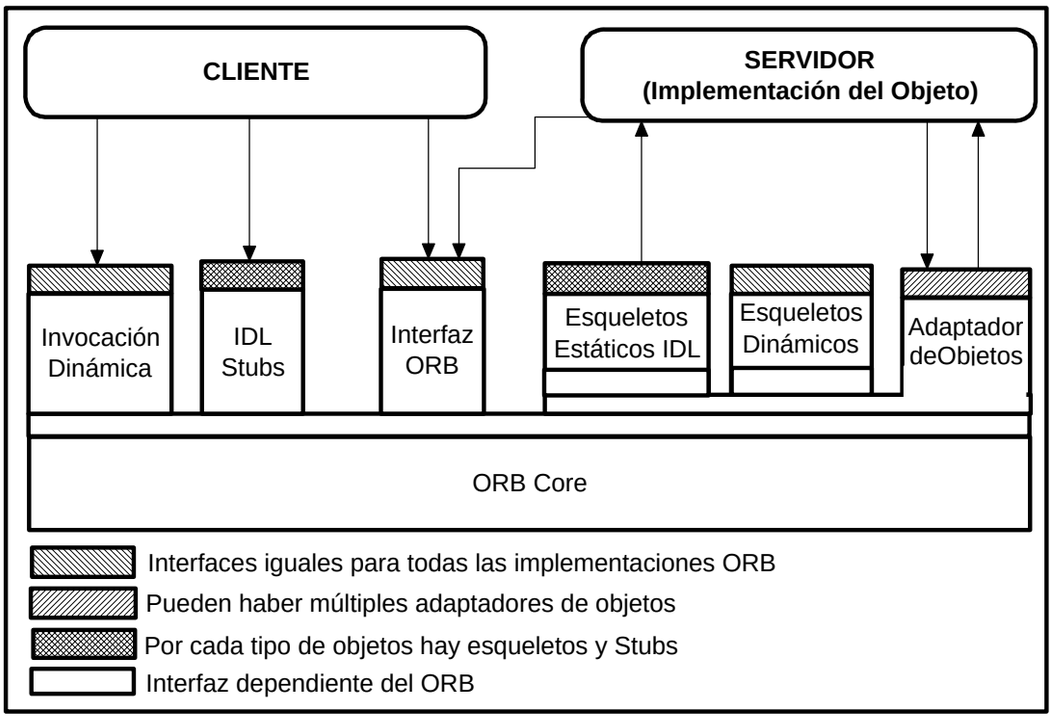


Figura 2.3-3: Estructura del ORB de CORBA

La interfaz del ORB

La interfaz ORB puede ser llamada tanto desde el cliente como desde la implementación del objeto. La interfaz provee un conjunto de funciones ORB que pueden ser invocadas directamente desde el cliente para recuperar alguna referencia a un objeto o por la implementación del objeto en sí. Estas interfaces son mapeadas a un lenguaje de programación específico dependiendo de las herramientas utilizadas. La interfaz ORB debe ser soportada por cualquier producto ORB.

Los stubs del cliente

El primer paso para invocar operaciones sobre un objeto remoto como si fuera local es la obtención de una referencia. Una vez que esta se ha

obtenido, el objeto puede invocar operaciones sobre esa referencia como si el objeto servidor fuera local. Sin embargo, esto no lo es. El cliente necesita ser enlazado con el “stub” para la interfaz definida en IDL que le permita que las invocaciones que éste realiza de forma local se transformen en requerimientos al ORB de ejecución de un método sobre el objeto remoto. En el objeto cliente, existe una operación (y un atributo) equivalente para cada uno de los definidos en el IDL de la clase a la que pertenece. Los stubs son también llamados “objetos proxy”. Los stubs son generados por el compilador de IDL para el lenguaje y el ORB específico que esté utilizando el cliente.

La interfaz de invocación dinámica (DII)

Por la descripción anterior, podemos decir que cualquier cliente debe ser enlazado con todos los stubs de los objetos servidores de los que pide servicios. Esto es así si se quiere utilizar invocación estática. Sin embargo, CORBA ofrece otro mecanismo más dinámico: el DII. Con esta interfaz, un objeto cliente puede invocar cualquier operación en objetos para los que no posee stubs. Una vez que ha obtenido la meta-información necesaria sobre el objeto sobre el que quiere realizar la invocación (como por ejemplo, saber sus métodos, número de argumentos de cada uno y tipo, etc.) puede construir un objeto del tipo Request (Requerimiento) que encapsula toda la información sobre la invocación de un método a un objeto y la respuesta que este devuelve (incluyendo las posibles excepciones que se lanzan durante la ejecución). Esto, junto con el sistema de meta-información, hace a CORBA ideal para entornos dinámicos en los que se incluyen en el sistema nuevos componentes en tiempo de ejecución y en los que los componentes se descubren unos a otros y son capaces de trabajar en organizaciones que no fueron pensadas en el momento de la creación de ninguno de ellos de forma independiente.

Hay que hacer notar que de cara al servidor, una llamada estática o una utilizando invocación dinámica aparecen totalmente iguales: el servidor no distingue si su cliente lo invoca desde un stub estático o utilizando una interfaz dinámica.

Los skeletons del servidor

Al igual que el cliente necesita sus stubs, el servidor necesita sus skeletons. Para cada interfaz que un servidor implementa necesita un skeleton. Estos son el equivalente para el servidor. Su función es transformar los requerimientos que el ORB realiza (que son a su vez producidas por invocaciones de clientes, ya sea estática o dinámicamente) en invocaciones sobre el objeto servidor o implementación real, además de retornar el resultado de la invocación (Fig. 1.2-3). Estos skeletons son también generados por el compilador IDL.

La interfaz de skeletons dinámicos (DSI)

El Dynamic Skeleton Interfaz permite a los servidores responder a requerimientos de invocación de forma dinámica (una vez más, no importa si el cliente generó este requerimiento de invocación de forma estática o dinámica: son equivalentes. “Dinámico” se refiere aquí al comportamiento del servidor frente a un requerimiento). Esto significa que el servidor puede responder a requerimientos interpretando en tiempo de ejecución el método invocado, los argumentos y su tipo, dándoles semántica dinámicamente. La información la obtiene de la interfaz ServerRequest. Esta técnica es útil para construir capas de software mínimas de componentes heredados, que simplemente pasan los requerimientos al componente encapsulado, obteniendo a cambio una integración en el ORB.

El adaptador de objetos (OA, Object Adapter)

El adaptador de objetos se encarga del manejo de objetos ya implementados. Así, se encarga de la instanciación, activación, desactivación de objetos implementados, el manejo de sus referencias, la conexión de una invocación con su correspondiente objeto servidor, etc. El estándar CORBA define dos adaptadores: BOA (Basic Object Adapter), genérico; y el POA (Portable Object Adapter), una estandarización más rígida y portable.

Como un ejemplo de la labor del BOA, existen cuatro políticas de la activación de objetos:

- Shared Server (Servidor Compartido). Si el proceso servidor contiene varios objetos. El BOA activa el proceso la primera vez que se realiza un requerimiento a alguno de los objetos que el proceso implementa.

- Unshared Server (Servidor no compartido). Si el proceso servidor contiene sólo un objeto. Para cada objeto nuevo que requiera de este objeto implementado, se crea uno nuevo para servirlo.

- Server-per-Method (Servidor por método). En esta política, el BOA activa un servidor para cada método invocado, que termina al finalizar la ejecución del método.

- Persistent Server (Servidor Persistente). Aquí, el servidor se ha iniciado por otro agente distinto al BOA. Lo único que hace éste es enviarle los requerimientos de los clientes.

Existe, además, otro adaptador específico para Sistemas de Gestión de Base de Datos Orientadas a Objeto (OODBMS), en el que, por ejemplo, por defecto todos los objetos son persistentes.

El Repositorio de Interfaces (IR)

El Repositorio de Interfaces es un servicio que ofrece objetos persistentes que representan la información de IDL de manera que ésta es accesible en tiempo de ejecución. Esta meta-información puede ser utilizada por los clientes pata construir invocaciones dinámicas. Este repositorio también es un lugar común donde se puede guardar información adicional sobre las interfaces, como información de depuración, etc.

Esta base de datos es actualizada por el compilador IDL, y ofrece al programador un conjunto de clases que describen la meta-información que

ésta posee y que permiten obtener esta información de una manera jerárquica.

El Repositorio de Implementaciones

El Repositorio de Implementaciones contiene información que permite al ORB localizar implementaciones de objetos. Esta información suele ser la del control de políticas, la información de instalación y, también otras informaciones adicionales, como información de depuración, control administrativo, reserva de recursos, seguridad, etc.

Comunicación entre ORB a vía protocolo IIOP

GIOP(General Inter-ORB Protocol) especifica la manera en que productos ORBs de diferentes vendedores pueden comunicarse entre sí. GIOP define un formato de mensajería y sintaxis de transferencia de datos estándar sobre un protocolo de transporte. No especifica un protocolo de comunicación particular. Es una especificación simple y neutral al protocolo que puede ser mapeada a diferentes protocolos de red.

Los mensajes GIOP están compuestos de requerimientos y respuestas en un modelo cliente/ servidor tradicional. Para invocar a un método, un cliente envía un requerimiento al servidor y luego espera por la respuesta. Una vez que el servidor recibe el requerimiento, lo procesa y envía la respuesta correspondiente al cliente.

La especificación GIOP define el mapeo de tipos de datos OMG IDL y referencias de objetos a una representación de red común conocida como CDR(*Common Data Representation*). CDR es un formato de datos que maneja la conversión de tipos de datos, ordenamiento de bytes y otras operaciones de bajo nivel para asegurar que el dato es transportado en un formato común, tal que las invocaciones ORB e información enviada entre maquinas sobre una red, sea el mismo objeto para ambas aplicaciones(cliente y servidor).

IIOP (Internet Inter-ORB Protocol) es una implementación específica de GIOP, define cómo los mensaje GIOP son mapeados al protocolo TCP/IP.

Para asegurar la interoperabilidad entre ORBs, la especificación CORBA define que todos los productos ORB deben soportar IIOP, ya sea nativamente o a través de un bridge.

IIOP es un protocolo de alto nivel que se preocupa de varios de los servicios asociados con los niveles que están sobre la capa de transporte, incluyendo traducción de datos, administración de buffer de memoria y administración de comunicación. También tiene la responsabilidad de direccionar los requerimientos a la instancia del objeto correcto dentro de un ORB. Una instancia de objeto se identifica por una IOR(Interoperable Object Reference), el cual está especificado en el GIOP y generado por el ORB.

Debido a que las referencias a objetos son manejadas de manera distinta por cada ORB, una IOR se utiliza para pasar referencias de objetos entre diferentes productos ORB. La aplicación cliente puede acceder a un objeto usando la IOR, la cual abstrae la implementación ORB de la aplicación cliente y de la implementación ORB usada por el servidor donde se encuentra el objeto CORBA.

2.3.4 Monitores de Procesamiento Transaccional

2.3.4.1 Conceptos

Los Monitores de Procesamiento Transaccional (Monitor TP) constituyeron una tecnología crítica por largo tiempo para las empresas orientadas al desarrollo de sistemas transaccionales. Los monitores TP fueron en un principio ampliamente empleados en computadores mainframes, productos de IBM y CICs. La tecnología del monitor TP nació hace 25 años atrás cuando Atlantic Power and Light crearon un ambiente de apoyo en línea para compartir en forma concurrente servicios de aplicaciones y recursos de información sobre ambientes de sistemas operativos batch o de tiempo compartido.

Un monitor TP ha sido concebido para gestionar procesos y coordinar programas garantizando la integridad, coherencia y seguridad de las aplicaciones.

Inicialmente los monitores TP fueron desarrollados como servidores multihilo para apoyar un gran número de terminales desde un único proceso central.

Proveen además, la infraestructura para construir y administrar sistemas de procesamiento transaccional con una gran cantidad de clientes y servidores.
Proveen servicios tales como:

- Servicios de presentación para simplificar la creación de interfaces de usuarios.
- Encolamiento persistente de los requerimientos de los clientes y respuestas de los servidores.
- Ruteo de los mensajes de los clientes a servidores.
- Coordinación a través del protocolo two-phase commit de las transacciones que involucran varios servidores [9].

Debido a que la esencia de un Monitor TP es el manejo de transacciones a gran escala, es conveniente definir el concepto de transacción:

Transacción

Esencialmente, una transacción es cualquier conjunto de operaciones que deben ser completados como una unidad. Si cualquier parte de la transacción no es completada, entonces toda la transacción debe ser restaurada a su estado original [10].
Una transacción es la implementación de una o más funciones del negocio basado en la asociación de las reglas de este, donde la transacción es completada sólo cuando todo lo solicitado en los requerimientos de las funciones se han completado según lo especificado en la regla del negocio.
Finalmente una transacción es un conjunto de acciones que deben respetar las propiedades ACID descritas a continuación.

Propiedades ACID

Una transacción, debe exhibir las propiedades ACID aparte de la derivada capacidad llamada rollback. A continuación explicaremos cada una de ellas.

- **Atomicidad.** Significa que una transacción es una unidad de trabajo indivisible. Todas las acciones de esta unidad de trabajo deben funcionar

para que la unidad funcione. Un ejemplo de ello, son los neutrones y protones de un átomo, ellas son partes que no pueden estar separadas una de otra. Es todo o nada.

- **Consistencia.** Una transacción después de su ejecución debe dejar el sistema en un estado coherente o en caso de problemas debe saber volver al estado existente antes de su ejecución.
  Una transacción siempre toma una base de datos desde un estado consistente y lo lleva a otro estado consistente; es decir, obedece a todas las reglas de integridad y movimientos hacia un nuevo estado consistente, de lo contrario permanece en su estado original.

- **Aislamiento.** Una transacción que se ejecuta no debe ser nunca afectada por otra transacción que se ejecuta al mismo tiempo, o sea, las transacciones deben ser tratadas como si ocurriesen secuencialmente en lugar de superponerse. Una transacción debe saber serializar el acceso a los recursos que comparte y garantizar que los programas que se ejecutan simultáneamente no van a poner en peligro la coherencia de los datos. Las modificaciones hechas por una transacción sobre los recursos compartidos no deben ser visibles para otras transacciones mientras no se ha ejecutado su commit.

- **Durabilidad.** Los cambios de una transacción son permanentes después de haberse llevado a cabo el commit. Las modificaciones deben poder sobrevivir a una caída del sistema.
  Desde un punto de vista práctico, la durabilidad generalmente significa que la información asociada con la transacción se almacena en un disco, en lugar de residir en la memoria del computador. En transacciones más críticas, para prevenir la posibilidad de pérdida de información debido a la falla de un disco, deben tomarse medidas fuertes que aseguren la durabilidad, tales como escritura en copias separadas de la información, separar discos o crear una entrada diaria de cual transacción puede ser recreada si es necesario.

2.3.4.2 Modelo de un Monitor TP

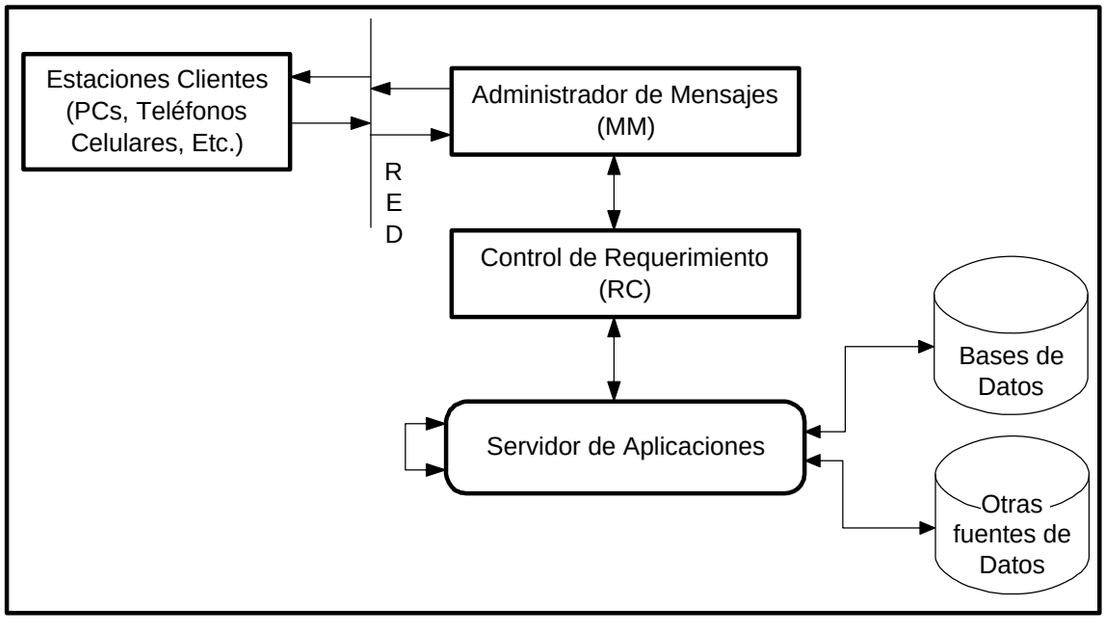
La función principal de un monitor TP es la de coordinar el flujo de transacciones requeridas entre estaciones de trabajo u otros dispositivos y programas de aplicación que puedan procesar dichos requerimientos [11]. Para lograr lo anterior, el monitor TP impone cierta estructura sobre los componentes de software de un sistema transaccional y ofrece funciones para soportar las actividades de los componentes.

La mayoría de las aplicaciones transaccionales han sido estructuradas de manera tal que puedan llevar a cabo los siguientes pasos:

1. Interactuar con la estación de trabajo del usuario para permitir el ingreso de los datos de entrada de una transacción específica.
2. Traducir los parámetros de entrada del paso anterior a un formato estándar para poder enviar el mensaje con la transacción requerida.
3. Comenzar la transacción.
4. Examinar el encabezado(header) del requerimiento para determinar su tipo.
5. Ejecutar la aplicación correspondiente al tipo de requerimiento entrante, el cual puede a su vez invocar a un DBMS(Sistema de Administración de Bases de Datos) y a otros programas de aplicación.
6. Realizar el commit de la transacción después que la aplicación ha terminado.
7. Enviar el resultado de la transacción a la estación de trabajo correspondiente.

Un monitor TP divide una aplicación en componentes para que sean éstos los responsables de realizar los puntos señalados anteriormente.

- Un Administrador de Mensajes(MM), el cual realiza los pasos (1),(2) y (7).
- Un Control de Requerimientos(RC), realiza los pasos (3),(4) y (6).
- Un Servidor de Aplicaciones(AS), realiza el paso (5), en colaboración la mayoría de las veces con DBMS.



2.3-4: Modelo de un Monitor de Procesamiento Transaccional.

En general los monitores transaccionales poseen varias instancias de MM, RC, AS y DBMS. Esas instancias siguen un paradigma de comunicación impuesto por el monitor TP: el componente MM se comunica con el componente RC, el cual se comunica con los Servidores de Aplicación y estos últimos se comunican con DBMS y con otros componentes de su categoría. Este paradigma de comunicación es consistente con el flujo de eventos en el procedimiento de los siete pasos indicados anteriormente. Al descomponer la aplicación de esta manera, el monitor TP puede simplificar la programación de aplicaciones mapeando esos componentes a procesos del sistema operativo, proveyendo adicionalmente soporte de comunicaciones entre los componentes. También debe proveer mecanismos de administración de sistemas para monitorear y controlar aspectos de rendimiento, fallas y seguridad.

Administrador de Mensajes

El Administrador de Mensajes (MM) realiza cuatro funciones principales: formatea requerimientos, maneja formularios, valida la entrada, despliega la salida y realiza chequeo de seguridad orientado al usuario.

Para aislar al componente Control de Requerimiento (RC) de las diversas interfaces provistas por las estaciones de trabajo (computadores personales,

teléfonos celulares, etc.), un MM traduce los *datos de entrada* que requieren la ejecución de una transacción a un *mensaje de requerimiento en formato estándar* o, simplemente, un requerimiento. De esta manera, el RC puede contar con una entrada en formato estándar. Esto hace a los programas RC independiente de los dispositivos de entrada, tal independencia provista por el MM es similar a la independencia de datos provista por un DBMS, para aislar las aplicaciones de una diversidad de formatos de base de datos físicas a través de un formato de base de datos estándar. El formato del requerimiento es definido por el monitor TP. Incluye un encabezado estándar, el cual es el mismo para todas las aplicaciones que usan el monitor TP y, un cuerpo del requerimiento, el cual es definido por la aplicación. El encabezado puede incluir la dirección de la estación de trabajo, el nombre del usuario y el nombre del tipo de requerimiento. El cuerpo del requerimiento debe incluir los parámetros de la transacción.

El **Administrador de Formularios** es el componente de un MM que tiene la responsabilidad de traducir desde el formato específico de la estación de trabajo al formato del requerimiento estándar. Cada formulario está compuesto por un conjunto de campos, cada campo tiene un conjunto de características, tales como nombre, un tipo de dato y una representación sobre el dispositivo de despliegue del cliente. El Administrador de Formularios también provee un compilador, el cual genera una tabla de traducción y una definición de registros a partir de la definición del formulario.

Un MM también es responsable de la Validación de la Entrada. Puede verificar que cada entrada es del tipo adecuado y que está en el rango de valores permitidos.

En un MM, el programador de aplicaciones escribe la definición de formularios y rutinas de validación de datos. El monitor TP hace el resto: compila la definición de formularios y hace la traducción en tiempo de ejecución de cada formulario en un requerimiento.

Por último, el MM realiza algunas funciones de seguridad. Autentifica cada usuario, verificando una password y pone el identificador del usuario en cada requerimiento que este emite. Puede también realizar control de acceso, el

cual verifica que un usuario determinado esté autorizado para utilizar una funcionalidad específica.

Control de Requerimientos

Cada requerimiento construido por un MM es transferido a un componente denominado Control de Requerimientos (RC). El RC tiene la responsabilidad de enviar el requerimiento a un Servidor de Aplicaciones (AS) que pueda procesar el requerimiento. El desarrollador de aplicaciones sólo tiene que proveer una tabla que relaciona cada tipo de requerimiento a un identificador del AS que pueda procesar el tipo de requerimiento.

El RC busca en el encabezado del requerimiento el tipo de requerimiento simbólico y, lo mapea en un identificador apropiado para el AS. El RC llama al AS que tenga el identificador, pasándole los parámetros que extrajo del requerimiento.

Mapeo RC-AS. El mapeo desde un tipo de requerimiento simbólico a un identificador AS podría ser dinámico. Esto es útil para la tolerancia a fallas, debido que permite al sistema remapear rápidamente un tipo de requerimiento a un identificador AS distinto, en caso que el original fallase.

Una tabla local al RC provee una forma fácil de implementar este mapeo dinámico. De haber más de una copia del RC, entonces cada copia puede tener una versión de esta tabla, conduciéndonos a un problema: si diferentes RC controlan diferentes tipos de requerimientos, entonces un requerimiento podría llegar a un RC que no puede manejarlo. Ante esta situación hay dos soluciones comunes:

- Cada MM conoce que RCs pueden manejar cada tipo de requerimiento. Cada MM está diseñado para enviar cada requerimiento R, a un RC que pueda manejar el tipo de requerimiento R.
- Cada RC conoce que RCs pueden manejar cada tipo de requerimiento. Un MM envía R a cualquier RC, el cual lo envía si es necesario a otro RC que maneje este tipo de requerimiento.

  Algunos sistemas soportan un Servicio de Nombres Global, que mapea nombres en pares atributo-valor, el cual es accesible desde cualquier nombre. Uno podría usar un servicio de nombres global para mapear

tipos de requerimientos a identificadores RC. Debido a que el nombre del servicio es globalmente accesible, cualquier MM o RC puede tomar la responsabilidad de reenviar cada requerimiento a un RC apropiado.

Enlace RC-AS. Para llamar a un AS, un RC debe utilizar el identificador del AS para poder enlazarlo. La naturaleza de esta unión la determina el sistema operativo y la arquitectura de comunicaciones.
También se puede utilizar un servicio de nombres global para almacenar el mapeo entre los tipos de requerimientos y los identificadores AS. Ya que el mapeo es accesible globalmente, los AS pueden accesarlo directamente, sin usar RCs como intermediarios. En este caso, el monitor TP no necesita distinguir entre RCs y ASs, es decir, la noción de RC desaparece.
Sin embargo, aún si el monitor TP no distingue entre RCs y ASs, las aplicaciones usualmente retienen esta distinción. Es decir, algunos ASs mapean tipos de requerimientos a identificadores ASs, y otros ejecutan el programa para este tipo de requerimiento. Esta estructura tiende a minimizar el número de enlaces, lo cual es importante por rendimiento en un sistema distribuido.

Servidor de Aplicaciones

Cada Servidor de Aplicación (AS) está compuesto de uno o más programas, los cuales proveen el código con la lógica del negocio a ejecutar, la cual típicamente accesa una o más bases de datos compartidas o cualquier otra fuente de datos.
Estos proveen balanceo de carga, poolling de hilo, reciclaje de objetos y la capacidad de recuperación automática frente a los problemas típicos de los sistemas.
Finalmente se puede mencionar que a través de estos servidores es posible crear aplicaciones completas construyendo muchos servicios de transacciones que puedan ser invocados desde el cliente.

2.3.4.3 Administración de Procesos de un Monitor TP

Una de las funciones de un monitor TP es la de definir una estrategia de administración de procesos para la creación y manejo de procesos para

MMs, RCs y ASs. Entenderemos por proceso, a una abstracción de sistema operativo la cual consiste en un espacio de direcciones, estado del procesador y un conjunto de recursos (una *tarea* en IBM MVS o *proceso* en sistemas operativos como UNIX o VAX/VMS). Hay varias estrategias de administración de procesos las cuales dependen de:

- Si los MMs, RCs y ASs se ejecutan juntos en un solo espacio de direcciones o separadamente en distintos espacios de direcciones.
- Si un proceso tiene uno o más de un hilo bajo su control, es decir, un solo hilo o múltiples hilos.

Hilo Único. Una estrategia simple de administración de procesos es la de crear un proceso por cada estación de trabajo. Cada proceso ejecuta una imagen que relaciona su MM, RCs y ASs. De esta manera, una llamada estándar interproceso puede ser usada por un MM para llamar a un RC y, por un RC para llamar a un AS, es decir, los procesos ejecutan un programa secuencial. Esta estructura de procesos por estación de trabajo es normalmente utilizada en sistemas de tiempo compartido, donde a cada estación de trabajo se le asigna un proceso único al momento en que el usuario hace uso del sistema. Desgraciadamente, este esquema no es escalable; cuando un sistema tiene un gran número de estaciones de trabajo, es ineficiente tener un proceso por cada cliente. La ineficiencia se origina de la sobrecarga en el sistema operativo con largas búsquedas de los descriptores de procesos en las tablas del sistema operativo; demasiado intercambio de contexto entre procesos; mucha memoria fija por proceso y el alto nivel de paginación de I/O.

Multi-Hilo. Uno puede evitar los problemas anteriores teniendo un solo proceso que maneje todas las estaciones cliente que estén conectadas a un nodo. Conceptualmente, cada estación de trabajo tiene un solo hilo que lo controla, pero comparte su dirección de espacio con todos los otros hilos en dicho proceso. Normalmente estos hilos son implementados por el monitor TP o por el sistema operativo. Cada hilo en un proceso debe tener un área de datos privada para datos locales. En caso que sea implementado por el sistema operativo, esta área normalmente está constituida por un stack

privado y un área de registro. Si es implementada por el monitor TP, está construida por un proceso local en una región de memoria e indexada por hilo.

2.3.4.4 Recuperación y Administración de Sistema

Los administradores de sistemas requieren herramientas en línea para monitorear y controlar todos los aspectos de un sistema de procesamiento transaccional activo, incluyendo rendimiento, fallas y seguridad. Estas herramientas recolectan datos y ajustan parámetros en varios componentes de los subsistemas. Esto es especialmente importante en grandes sistemas distribuidos, en los cuales la complejidad y el control distribuido es muy difícil de manejar. Los administradores de sistemas también necesitan herramientas fuera de línea que le permitan probar las primeras versiones de una aplicación y para analizar los datos producidos por
las herramientas de monitoreo, esto permite efectuar una planificación respecto del rendimiento, posibles fallas y aspectos de seguridad.

Un monitor TP provee operaciones de administración de sistema para manejar el conjunto de procesos MM, RC y AS. Para hacer esto, el monitor TP mantiene una descripción de la configuración de procesos en el sistema. Esta descripción incluye las estaciones de trabajo y formularios atachados a cada MM, las características de seguridad de los usuarios, el conjunto de tipos de requerimientos enrutados por cada RC, el conjunto de programas administrados por cada AS, etc. En un sistema distribuido, también incluye el nombre de los nodos sobre los cuales se ejecuta cada proceso. De esta manera un administrador de sistemas puede crear y distribuir procesos, moverlos entre nodos y alterar el conjunto de formularios y programas usados por cada proceso.

El monitor TP puede medir el rendimiento de los sistemas que están en ejecución y ofrecer esta información al administrador de sistemas en términos orientados a la aplicación: tasas de transacción, tiempos de respuesta, etc. El administrador de sistemas puede utilizar esta información para ajustar la configuración o para mejorar el tiempo de respuesta.

El conocimiento del administrador de sistemas respecto de la configuración MM-RC-AS es útil para poder manejar las fallas. Si un nodo falla, el monitor TP puede recrear los MM del nodo sobre otro nodo que tenga acceso al

mismo conjunto de estaciones de trabajo y pueda crear sesiones entre las estaciones clientes y los nuevos MM. Puede también recrear los ASs y RCs de los nodos con falla sobre otro nodo que pueda ejecutar los programas apropiados y tenga la capacidad disponible para ejecutar los procesos. Usando su descripción de la configuración, el monitor TP puede realizar esas acciones sin interacción humana.

La abstracción de una transacción y requerimientos encolados ayuda a que la recuperación sea transparente. Cuando un proceso falla, las transacciones que fueron ejecutadas en el proceso son canceladas. Después que el monitor TP recupera el proceso con errores (posiblemente sobre otro nodo), el requerimiento que corresponde a las transacciones canceladas son automáticamente retomadas. Si esta recuperación es lo suficientemente rápida, el usuario de la estación cliente ve esta falla como si tan solo fuese una baja en el tiempo de respuesta.

2.3.4.5 Arquitectura de Monitores TP

A continuación se describirán cuatro modelos de arquitectura del monitor TP:

- Modelo proceso por cliente. En lugar de una sesión de login individual por estación de trabajo, el proceso servidor se comunica con la estación de trabajo, maneja la autenticación y ejecuta acciones. En esta arquitectura los requerimientos de memoria son altos y las multitareas sobrecargan la CPU con el cambio de contexto entre procesos. Ver la figura 2.3-5.

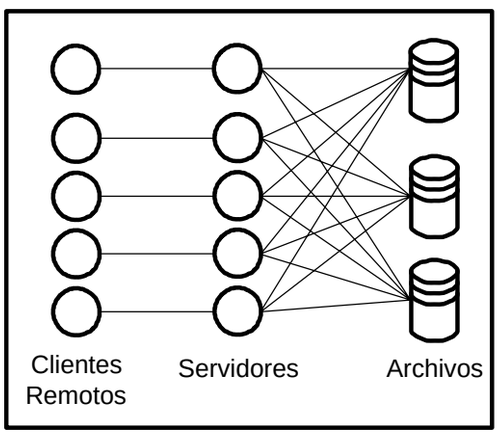


Figura 2.3-5. Modelo de proceso por cliente.

- Modelo proceso único. En este modelo todas las estaciones remotas se conectan a un único proceso servidor. Se usa en ambientes cliente/servidor, donde el proceso servidor es multihilo; lo que permite un bajo costo para el cambio de hilo. Esta arquitectura no está preparada para base de datos distribuidas o paralelas. Ver la figura 2.3-6.

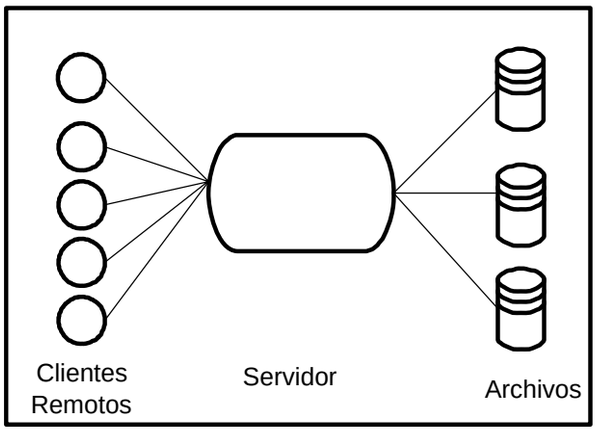


Figura 2.3-6. Modelo de único proceso.

- Modelo muchos servidores único ruteo. Los múltiples procesos de los servidores de aplicación accesan una base de datos común; la comunicación de los clientes con la aplicación es a través de un único proceso de comunicación que rutea los requerimientos donde corresponda. Esta arquitectura tiene servidores con procesos de multihilo y se ejecuta sobre bases de datos paralelas o distribuidas. Ver figura 2.3-7.

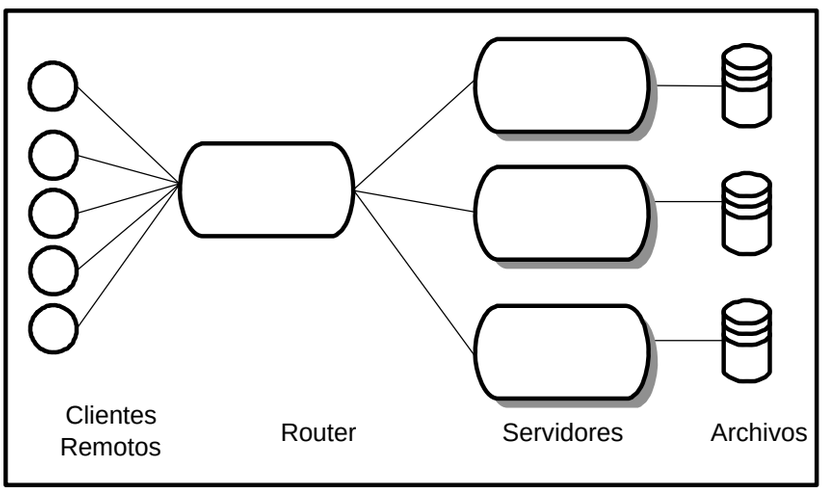


Figura 2.3-7. Modelo de muchos servidores y un único router.

- Modelo muchos servidores y muchos routers. En este modelo existe continuación entre múltiples procesos y múltiples clientes. Los procesos de continuación del cliente interactúan con los procesos de un router que envía sus requerimientos al servidor apropiado. Ver la figura 2.3-8.

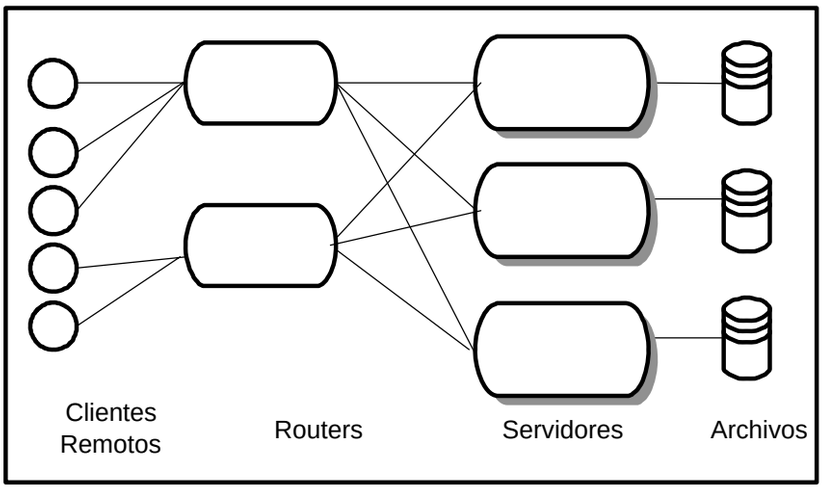


Figura 2.3-8. Modelo de muchos servidores y muchos routers.

Características de un Monitor TP

- Multiplexión de Base de Datos : El beneficio real de un servidor de aplicaciones es la capacidad de multiplexar y manejar las transacciones, lo que resulta en la reducción de la cantidad de conexiones y procedimientos almacenados que se ubican en grandes sistemas o bases de datos. Con servidores de aplicaciones en la arquitectura, es posible aumentar la cantidad de clientes sin aumentar el tamaño del servidor de la base de datos. Un cliente invoca un servicio transaccional que reside en un monitor TP, y estos servicios pueden compartir las mismas conexiones del servidor de base de datos.

- Balanceo de Carga: Ocurre cuando la cantidad de requerimientos entrantes sobrepasa el umbral de procesos compartidos que el sistema es capaz de manipular, en ese instante se inician automáticamente otros procesos, este es el concepto del balanceo de carga. Algunos servidores de aplicaciones pueden distribuir los procesos de carga sobre varios

servidores al mismo tiempo o distribuir el procesamiento sobre varios procesadores en ambientes de multiproceso.

Las características de balanceo de carga de los servidores de aplicación también permite a los mismos manejar prioridades en las transacciones. Los servidores de aplicaciones son capaces de manipular las prioridades definiendo “clases”. Clases de alta prioridad que activan los procesos prioritarios. Como regla, los desarrolladores usan clases de servidores de alta prioridad para encapsular funciones de corta ejecución y alta prioridad. Los procesos de baja prioridad (tales como procesos batch) se ejecutan dentro de clases de servidores de baja prioridad. Por otro lado, los desarrolladores pueden asignar prioridades por tipo de aplicación, para el manejo de recursos requeridos por transacción, tiempos de respuesta altos y bajos y, la tolerancia a fallas de una transacción. Los desarrolladores pueden también controlar la cantidad de procesos o hilos (threads) disponible por cada transacción definiendo la cantidad de parámetros.

- Tolerancia a Fallas: Los servidores de aplicación fueron construidos desde sus inicios para proveer ambientes de desarrollo de aplicaciones robustas con la capacidad de recuperarse desde cualquier problema relacionado al sistema. Los servidores de aplicaciones proveen alta disponibilidad empleando sistemas redundantes. Las transacciones trabajan a través del protocolo two-phase commit para asegurar que las transacciones se completen. El two-phase commit también asegura que las transacciones confiables puedan ser ejecutadas en dos o más recursos heterogéneos. En eventos de fallas poderosas, por ejemplo, los servidores de aplicaciones alertan a todos los participantes que la transacción en particular (servidor, colas, clientes, etc) del problema. Algunos o todos los implicados en el trabajo del punto anterior son capaces de hacer rollback y así, el sistema retorna a su estado de “pretransacción”, limpiando cualquier desarreglo que pueda haber ocurrido.

- Comunicaciones: Las aplicaciones de servidores proveen un buen ejemplo de middleware que usan middleware, los cuales incluyen

mensajes brokers. Las aplicaciones de servidores se comunican de diversas formas, incluyendo RPC (especializados para servidores de aplicaciones y llamadas “transaccionales RPCs”), comunicaciones entre procesos y MOM.

3 DESARROLLO DE LA METODOLOGÍA

3.1 DEFINICIÓN DE TUXEDO

Conceptualmente el sistema BEA Tuxedo es un Middleware de Procesamiento Transaccional, capaz de distribuir aplicaciones a través de múltiples plataformas, bases de datos y sistemas operativos usando comunicaciones basadas en mensajes. Adicionalmente posee la capacidad de procesamiento transaccional distribuido.

Por otro lado, Tuxedo mejora el modelo cliente/servidor de dos capas, al permitir la separación entre los clientes y los sistemas de bases de datos a través de la inclusión de una capa intermedia que posee la lógica de negocios según se muestra en la figura 3.1-1.

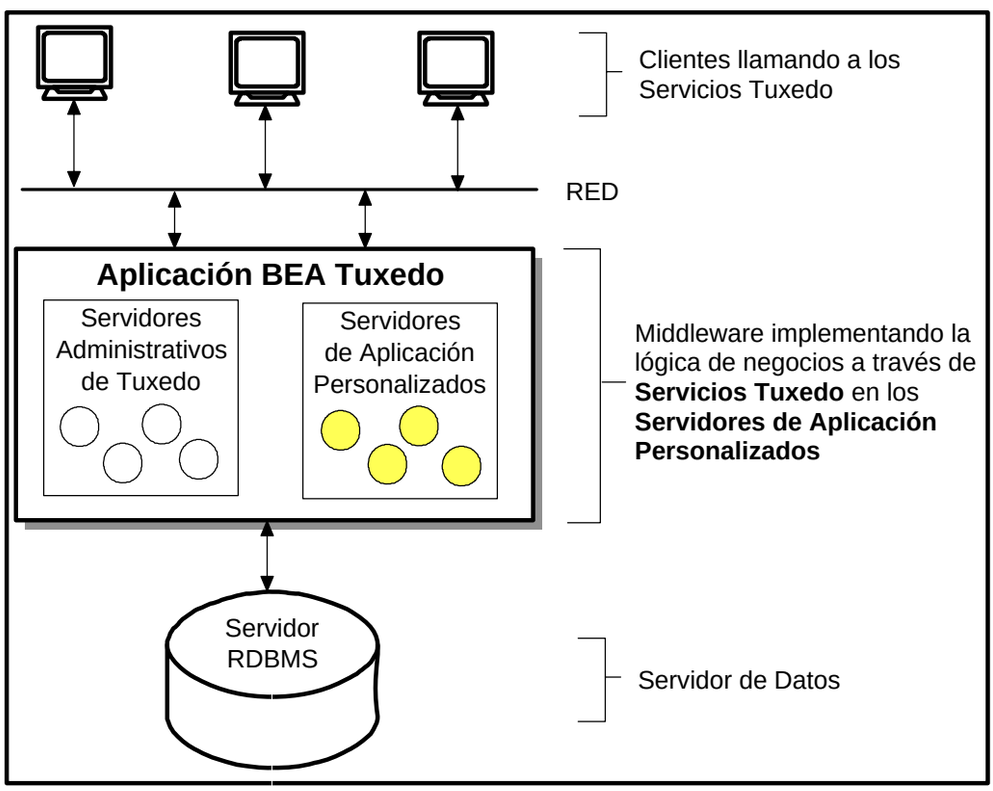


Figura 3.1-1: Concepto de BEA Tuxedo.

En una arquitectura cliente/servidor, los **clientes** (programas que representan a los usuarios que necesitan servicios) y los **servidores** (programas que proveen los servicios) son objetos lógicos separados que se comunican a través de una red para realizar tareas en conjunto. Un cliente

hace un requerimiento a un servicio y recibe una respuesta al requerimiento, por su parte, el servidor recibe y procesa un requerimiento y devuelve la respuesta. En este contexto, entenderemos por **servicio**, una función que reside en el servidor y que implementa una lógica de negocio específica, con un conjunto de parámetros de entrada y salida bien definidos.

3.1.1 Características

- Protocolos Asimétricos, hay una relación muchos a uno entre clientes y servidores. Los clientes siempre inician un diálogo de requerimiento hacia un servicio. Los servidores esperan pasivamente por los requerimientos de los clientes.
- Encapsulación de Servicios, cuando llega un mensaje requiriendo un servicio, el servidor sabe como resolverlo. Los servidores pueden ser actualizados sin afectar a los clientes, siempre que se mantenga sin cambios la interfaz del mensaje publicado.
- Integridad, el código y datos asociados a un servidor se mantienen centralizadamente, lo cual disminuye los costos de mantención y aumenta la protección de los datos compartidos. Del mismo modo, los clientes se mantienen como unidades independientes.
- Transparencia de Localización, el servidor es un proceso que puede residir sobre la misma máquina en la que reside el cliente, o sobre una distinta a través de una red. El modelo cliente/servidor normalmente oculta la localización de un servidor de sus clientes redireccionando los requerimientos donde corresponda. Un programa puede ser un cliente, un servidor o ambos.
- Intercambio basado en mensajes, los clientes y servidores son procesos débilmente acoplados que pueden intercambiar requerimientos de servicios y réplicas usando mensajes.
- Diseño extensible y modular. El diseño modular de aplicaciones cliente/servidor permite que la aplicación sea tolerante a fallas. En un sistema tolerante a fallas, las fallas pueden ocurrir sin causar una paralización de la aplicación completa. En una aplicación cliente/servidor tolerante a fallas, uno o más servidores pueden fallar sin detener al sistema completo siempre y cuando, los servicios ofrecidos por los

servidores con problemas estén siendo provistos adicionalmente por otros servidores de respaldos. Otra ventaja de la modularidad es que una aplicación cliente/servidor puede determinar automáticamente si aumenta o disminuye la carga del sistema activando o desactivando uno o más servicios o servidores.

- Independencia de la Plataforma, el ideal de un software cliente/servidor es que sea independiente de las plataformas de hardware y de sistema operativo, permitiendo combinar diferentes plataformas de clientes y servidores. Los clientes y servidores pueden ser distribuidos sobre diferente hardware utilizando distintos sistemas operativos, optimizando de este modo, el tipo de trabajo que realizan.
- Código Reutilizable, los servicios pueden ser usados en diferentes servidores.
- Escalabilidad, los sistemas cliente/servidor puede ser escalados horizontalmente y verticalmente. El escalamiento horizontal significa agregar o quitar estaciones de trabajo cliente con un impacto despreciable en el rendimiento. El escalamiento vertical significa migrar a una máquina servidora de mayor capacidad o agregar varias máquinas servidoras.
- Separación de la funcionalidad Cliente/Servidor, la cual es una relación entre procesos ejecutándose sobre la misma máquina o sobre máquinas distintas. Un servidor es un proveedor de servicios y un cliente es un consumidor de servicios. El modelo cliente/servidor provee una clara separación de funciones.
- Recursos Compartidos, un servidor puede proveer servicios a varios clientes al mismo tiempo y regular sus accesos a los recursos compartidos.

3.2 COMPONENTES DE TUXEDO

En esta sección se definirán los elementos básicos que componen una Aplicación Tuxedo. Entenderemos el término Aplicación como un conjunto de programas y recursos que permiten realizar funciones de negocio. Una aplicación Tuxedo adicionalmente incluye los recursos utilizados por el sistema Tuxedo para soportar dichos programas.

Una aplicación distribuida Tuxedo está compuesta de las siguientes partes:

- Programas Clientes.
- Rutinas de Servicios.
- Programas Servidores.
- Recursos usados por la lógica de negocio como bases de datos, colas de aplicación y eventos.
- Recursos del sistema operativo.
- Recursos de hardware.

Cabe señalar en forma especial que el sistema Tuxedo provee funciones administrativas a través de un conjunto de utilitarios, programas servidores y recursos que almacenan la información necesitada y producida por el sistema. También, el sistema Tuxedo utiliza recursos del sistema operativo, tales como, memoria compartida y espacio de disco necesarios para la operación del sistema. Por último, las diferentes partes de la aplicación pueden ser distribuidas sobre un conjunto de máquinas con diferentes sistemas operativos.

El sistema Tuxedo utiliza el concepto de **Información Base para la Administración Tuxedo** (TMIB) o simplemente MIB, para referirse a una base de información mantenida en memoria (Bulletin Board) que contiene información estática y dinámica que Tuxedo maneja en términos de clases genéricas y atributos:

- Clientes.
- Servidores.
- Servicios.
- Procedimientos.
- Ruteo.
- Colas
- Máquinas.
- Dominios.

La figura que sigue a continuación muestra los elementos que componen una aplicación Tuxedo.

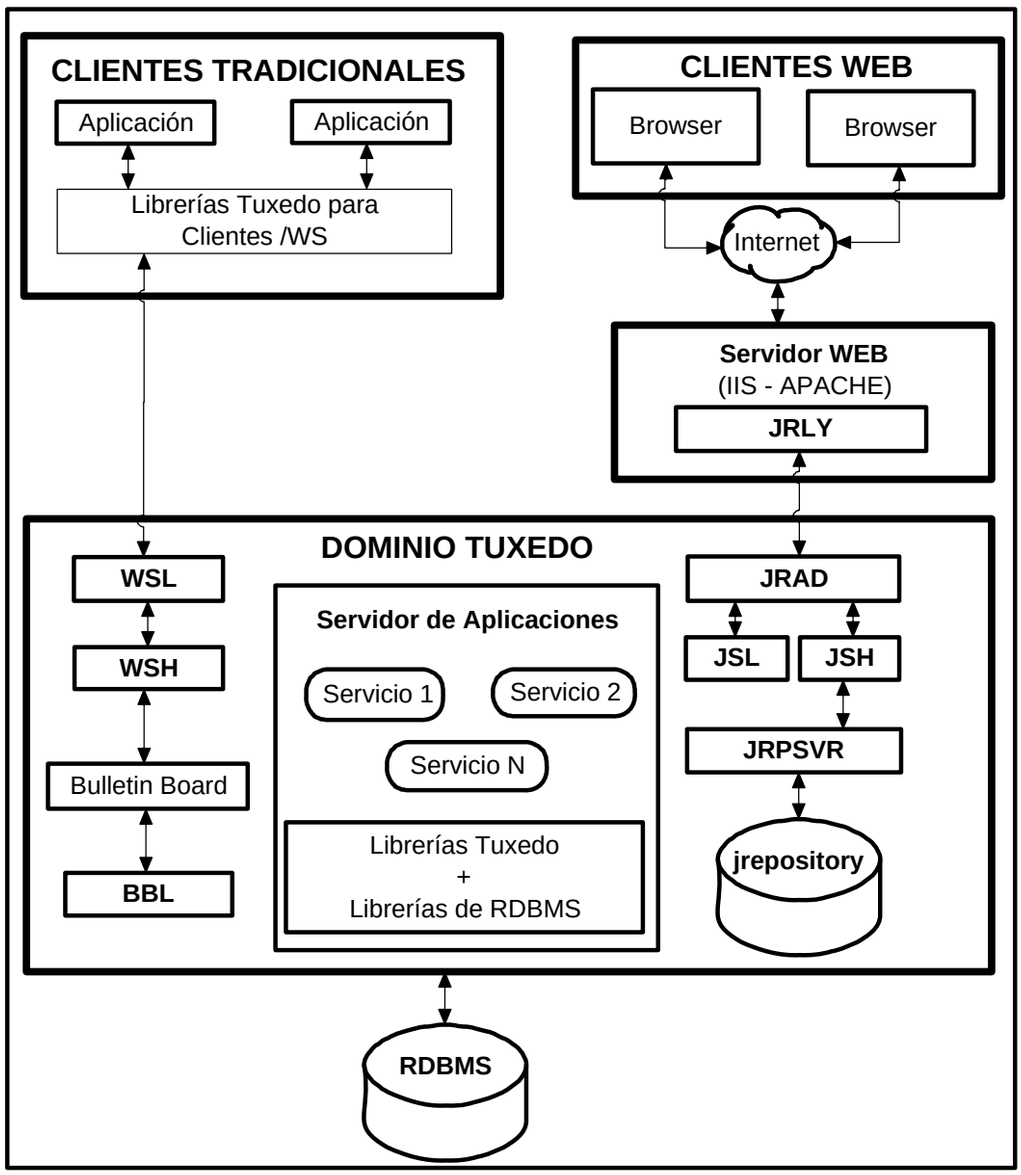


Figura 3.2-1 Elementos de una Aplicación Tuxedo.

El primer concepto que se debe rescatar de la figura, es el que define a un **Dominio** como una aplicación administrada en forma autónoma, cuyas características se especifican en un archivo de configuración **UBB**. Este archivo de texto está compuesto de varias secciones, cada una de las cuales define aspectos específicos del dominio. El conjunto de secciones posibles que puede contener un archivo UBB son:

Nombre Sección	**Descripción**
RESOURCES	Esta sección especifica aspectos generales del dominio, como nombre, número máximo de servidores,

	número máximo de servicios, etc.
MACHINES	Esta sección permite establecer parámetros lógicos asociados a una máquina física.
GROUPS	Sección que define los grupos que contendrán los servidores.
NETGROUPS	Esta sección define los grupos de red disponibles en la aplicación.
NETWORK	Esta sección describe la configuración de la red.
SERVERS	Define los servidores de aplicación que componen el dominio. Se especifican en esta sección los servidores administrativos y los servidores con la lógica de la aplicación.
SERVICES	Define los servicios que componen el dominio. Aquí se nombran los servicios con la lógica de la aplicación.
ROUTING	Esta sección define el ruteo dependiente de los datos.

El sistema Tuxedo no utiliza este archivo directamente, más bien se basa en una versión compilada (binaria) de éste.

El **Bulletin Board** es una colección de estructuras de datos compartidas en memoria diseñada para mantener la información de control de la aplicación Tuxedo mientras está en ejecución. Contiene información acerca de los servidores, servicios, clientes y transacciones pertenecientes a la aplicación Tuxedo. El Bulletin Board es replicado en cada una de las máquinas lógicas nativas en la aplicación.

El **BBL** (Bulletin Board Liaison) es el nombre dado a un servidor de administración Tuxedo que maneja el Bulletin Board. Algunas veces el Bulletin Board en sí es llamado BBL. Cada nodo Tuxedo tiene un Bulletin Board y un BBL.

En el contexto de manejo de clientes tradicionales es importante entender el rol que cumplen los servidores administrativos WSL y WSH que se describen a continuación.

Un **WSL** (Workstation Listener) es un proceso administrativo provisto por el sistema Tuxedo, responsable de representar un único punto de contacto con los clientes de tipo workstation (WS, definido más adelante). Este también administra la distribución de las conexiones workstation hacia los WSH,

iniciando nuevas instancias de estos si fuese necesario. Este proceso reside dentro del dominio administrativo de la aplicación.

El **WSH** (Workstation Handlers), es un cliente sustituto provisto por Tuxedo, responsable de manejar un conjunto de conexiones de clientes. Los WSH son activados dinámicamente por el WSL. Este proceso reside dentro del dominio administrativo de la aplicación. Los WSH son registrados en el Bulletin Board local de Tuxedo como clientes.

Para los clientes WEB hay que considerar que existen elementos adicionales dentro de la arquitectura ya que estos no se conectan directamente al dominio Tuxedo, sino a un WEB Server y es este, el que se conecta al dominio a través de un proceso intermedio llamado JRLY; de ahí en adelante el esquema es similar al utilizado en los clientes tradicionales, pero los servidores en cuestión son JRAD, JSL y JSH.

El **JRLY** (Jolt Relay) es un programa cuyo objetivo es recibir requerimientos de un servidor WEB y redireccionarlos a un proceso del dominio Tuxedo denominado JRAD. Normalmente el JRLY se ejecuta en la misma máquina del servidor WEB la cual es distinta a la del dominio Tuxedo. Esta separación se hace por razones de seguridad en la que hay involucrados firewall.

El **JRAD** (Jolt Relay Adapter) es un servidor provisto por el sistema Tuxedo que no exporta servicios, responsable de gestionar la comunicación con JRLY. Este también administra la distribución de las conexiones hacia los JSL y JSH. Este proceso reside dentro del dominio administrativo de la aplicación.

En el contexto de una aplicación WEB, existe un repositorio con las especificaciones de los servicios Tuxedo denominado **jrepository**, el cual puede contener un subconjunto o el conjunto total de servicios del dominio. Por lo tanto, un cliente WEB podrá llamar sólo a los servicios que estén inscritos en el repositorio. El responsable de administrar este repositorio es un servidor provisto por Tuxedo denominado **JREPSVR**.

3.3 TIPOS DE MENSAJES TUXEDO

Todos los mensajes transmitidos entre aplicaciones Tuxedo (clientes y servidores) deben usar un tipo de mensaje o buffer conocido. Actualmente los principales tipos de Buffer que maneja Tuxedo son:

- STRING
- VIEW y VIEW32
- FML y FML32
- CARRAY
- X_OCTET

De este conjunto, nos concentraremos principalmente en los mensajes FML32 por cuanto ofrecen el mayor grado de flexibilidad y porque además es el esquema que usa Tuxedo en forma interna para alguna de sus tareas. Además se definirán los dos primeros grupos por ser los más usados.

3.3.1 Tipo STRING

Los STRING buffers se definen como una sola cadena de caracteres terminada en NULL. Este tipo de buffer es convertido automáticamente por Tuxedo cuando se trata de distintas plataformas. La principal desventaja de este tipo de Buffer es que no permite RUTEO dependiente de los datos. La otra desventaja es que el desarrollador está obligado a analizar (parser) los mensajes si desea identificar campos específicos.

Consideremos un mensaje que se envía al servidor con tres parámetros de entrada (Rut, Nombre, Dirección), en este caso la forma del mensaje sería:

Mensaje 1: "10633497-8|Jose Catalán|Providencia 777"

Mensaje 2: "100-7|Pedro Lemus|Miraflores 388 Piso 4"

Observe que es necesario establecer un orden en campos que no puede ser alterado, porque el servidor necesitará analizar el mensaje para obtener cada parámetro del String. Además es necesario un Caracter que haga las veces de separador de campo, en el ejemplo se utilizó el pipe |.

3.3.2 Tipo FML y FML32

Los FML Buffer (Field Manipulation Language) están diseñados para utilizar un conjunto de funciones en C para poder acceder a los datos y no estructuras como lo hacen las VIEWs que serán descritas en el punto 3.3.3. Los FML Buffers también permiten conversión automática de datos entre distintas máquinas y Ruteo dependiente de los datos.

Hay dos versiones de FML Buffers: FML y FML32 (16 y 32 bits). Lo cual hace necesaria una distinción a nivel de nombres de funciones y utilitarios, en FML32 se agrega un "32" al final de los nombres de sus análogos en FML. Por ejemplo: la función Fchg cambia el contenido de una field en FML, mientras que Fchg32 hace lo mismo en FML32.

Para poder utilizar este tipo de buffer se necesita seguir un conjunto de pasos que se indican a continuación:

- Definir una tabla de fields que corresponde a un archivo de texto con un formato específico, que contiene los campos que se desean manejar en los mensajes.
- Ejecutar el comando Tuxedo mkfldhdr32 para crear un archivo *.h que contiene los Identificadores de Fields que utiliza Tuxedo para reconocer los campos dentro de un Buffer FML32. El concepto de Identificador de Field se verá más adelante ya que es básico para poder trabajar con esta clase de mensajes.
- Use #include para incluir los archivos de cabecera en los programas clientes y programas servidores que lo requieran.
- En runtime tanto los programas clientes como servidores necesitan las siguientes variables de ambiente para obtener nombres e identificadores de field:

Nombre Variable	Significado
FLDTBLDIR32	Es el directorio donde se encuentran los archivos que contienen las tablas. De haber más de un directorio se separarán por ; en plataformas Windows y por : en plataformas UNIX
FIELDTBLS32	Es una lista separada por comas con los nombres de los archivos con la especificación de las tablas de fields. De haber más de una tabla se separarán por

	comas , Ejemplo: FIELDTBLS32=tabla1,tabla2,tabla3

3.3.2.1 Especificación de la Tabla de Fields FML32

Cada tabla tiene el siguiente formato:

- Líneas que comienzan con # o en blanco se ignoran.
- Líneas que comienzan con $ se copian íntegramente al header *.h generado.
- Líneas que comienzan con *base indican un desplazamiento que se suma al numero asignado al campo.
- Una lista de fields o campos.

Las columnas en cada línea de especificación pueden ser separadas por blancos o tabs.

Ejemplo de un archivo de especificación de tabla de fields para la tabla de nombre TblPersona, con una base de 3000:

```
#   línea de comentario
*base 3000
# name        Number   type     flags     comments
FldRut          0      string     -       Rut de la persona
FldNombre       1      string     -       Nombre
FldEdad         2      long       -       Edad de la persona
FldMsg          3      string     -       Mensaje
FldApellido     4      string     -       Apellido
```

Ahora se define la misma tabla, pero el número del campo está puesto explícito (sin el campo *base)

```
#   línea de comentario
# name        number   Type     flags     comments
FldRut          3000   String     -       Rut de la persona
FldNombre       3001   String     -       Nombre
FldEdad         3002   long       -       Edad de la persona
FldMsg          3003   string     -       Mensaje
FldApellido     3004   string     -       Apellido
```

Ambas tablas generan el mismo resultado.

Estructura de las columnas:

COLUMNA	SIGNIFICADO
name	Nombre del Field.
number	Número de Field, asignado por el usuario y único para un tipo de dato específico.
type	Es el tipo del field, valores posibles: char, string, short, long, float, double, carray.
flag	No se usa actualmente. Uso futuro.
comments	Comentario descriptivo del field.

3.3.2.2 Generación del Archivo de Cabecera

Para generar el archivo de cabecera y poder incorporarlo a los programas clientes y servidores se debe ejecutar el siguiente comando:

```
mkfldhdr32 [-d directorio_destino] [NombreTabla1 NombreTabla2 ....]
```

Donde directorio_destino es el directorio donde se dejarán los archivos generados. Si no se especifica se dejan en el directorio actual. NombreTabla-i es el nombre de la tabla de entrada, si no se especifica el comando usará la variable de ambiente FIELDTBLS32 para generar un archivo por cada tabla en la variable de ambiente.
En nuestro ejemplo, el archivo de cabecera TblPersona.h quedaría con la siguiente estructura luego de ejecutar el comando *mkfldhdr32 TblPersona*.

```
/*      fname                   fldid           */
/*      -----                   -----           */
#define FldRut          ((FLDID)43960)      /* number: 3000  type: string */
#define FldNombre       ((FLDID)43961)      /* number: 3001  type: string */
#define FldEdad         ((FLDID)11194)      /* number: 3002  type: long   */
#define FldMsg          ((FLDID)43963)      /* number: 3003  type: string */
#define FldApellido     ((FLDID)43964)      /* number: 3004  type: string */
```

Hay que observar que el campo generado fldId está en función del tipo de dato (string) y del número especificado por el usuario (3000).
En términos de programación, cuando se llaman a las funciones FML32 desde un programa en C es necesario referirse a los campos a través de los nombres dados.

3.3.2.3 Uso de los Archivos Generados

En los programas clientes y servidores se deberán utilizar estos archivos de cabecera a través de una sentencia #include.
Para aquellos ambientes de desarrollo (Visual Basic, Delphi, Centura) de clientes que no soportan los #include, se deben utilizar las tablas originales vía la variable de ambiente FIELDTBLS32 y poder así acceder a los identificadores field por medio de APIs especiales.
Para incluir los llamados a las API's FML32 en los programas de aplicación en lenguaje C, es necesario incluir las librerías asociadas a este tipo de buffer.
En el caso de COBOL hay que hacer conversiones especiales ya que no soporta directamente las APIs FML32.

3.3.2.4 Tipos de datos e Identificación de Field

Los tipos de datos que se pueden manipular en un Buffer FML32 son los siguientes:

- short
- long
- char
- float
- double
- string
- carray

Estos tipos están definidos en un archivo de cabecera de Tuxedo como sigue:

```
#define FLD_SHORT        0      /* short int                    */
```

```
#define FLD_LONG        1    /* long int                    */
#define FLD_CHAR        2    /* character                   */
#define FLD_FLOAT       3    /* single-precision float      */
#define FLD_DOUBLE      4    /* double-precision float      */
#define FLD_STRING      5    /* string - null terminated    */
#define FLD_CARRAY      6    /* character array – binario*/
```

Para un mejor entendimiento del tema definiremos el concepto de **Identificador de Field,** básicamente es la definición de una constante que representa un campo en un Buffer FML32 y que lleva implícito en su definición el tipo.

Lo que hace Tuxedo es transformar un Número de Field dado por el Usuario a una constante que lleva además implícito el tipo de dato, el rango de valores posibles del usuario es [101 - 33.554.431].

Tabla de valores Mínimos para cada tipo de dato:

Tipo de Dato Tuxedo	**Definición Tuxedo**	**Número Field (Usuario)**	**Identificador Field (Usado por Tuxedo)**	**Representación Binaria Identificador Field (Tuxedo)**
FLD_SHORT	0	101	101	000**00000000000000000001100101**
FLD_LONG	1	101	33554533	001**00000000000000000001100101**
FLD_CHAR	2	101	67108965	010**00000000000000000001100101**
FLD_FLOAT	3	101	100663397	011**00000000000000000001100101**
FLD_DOUBLE	4	101	134217829	100**00000000000000000001100101**
FLD_STRING	5	101	167772261	101**00000000000000000001100101**
FLD_CARRAY	6	101	201326693	110**00000000000000000001100101**

Tabla de valores Máximos para cada tipo de datos:

Tipo de Dato Tuxedo	**Definición Tuxedo**	**Número Field (Usuario)**	**Identific. Field (Usado por Tuxedo)**	**Representación Binaria Identificador Field (Tuxedo)**
FLD_SHORT	0	33554431	65535	000**00000000111111111111111**

FLD_LONG	1	33554431	33619967	001**00000000011111111111111111**
FLD_CHAR	2	33554431	67174399	010**00000000011111111111111111**
FLD_FLOAT	3	33554431	100728831	011**00000000011111111111111111**
FLD_DOUBLE	4	33554431	134283263	100**00000000011111111111111111**
FLD_STRING	5	33554431	167837695	101**00000000011111111111111111**
FLD_CARRAY	6	33554431	201392127	110**00000000011111111111111111**

Los valores que finalmente utilizan las APIs Tuxedo corresponden al valor de la columna Identificador Field, que en el fondo es el Número de Field transformado.

Otra cosa importante es que para un mismo Número de Field y distintos tipos, Tuxedo generará distintos Identificadores de Field.

Cuando se utiliza este tipo de Buffer en un mensaje Tuxedo, este tiene la siguiente forma genérica:

fldid	largo	dato	fldid	largo	dato	fldid	largo	dato	……

3.3.3 Tipo VIEW y VIEW32

Los Buffers tipo VIEW32 permiten intercambiar mensajes basados en estructuras en C. Los Buffers tipo View32 permiten hacer conversión automática y Ruteo dependiente de los datos. Hay dos tipos de VIEW Buffer: aquellos usados en combinación con FML32 Buffers (FML sobre Cobol) y los que son manejados independiente de los FML32 Buffer.

Además hay dos versiones de VIEW Buffers: View y View32. VIEW usa short integer para almacenar el tamaño de los campos, mientras que VIEW32 usa long integer. Como consecuencia de esto, se puede llegar a almacenar campos de VIEW32 de hasta 2MBG como máximo.

Otra cosa que hace distintos a las funciones y utilitarios de VIEW32 es que poseen un "32" agregado al final de los nombres de sus análogos en VIEW. Por ejemplo: el utilitario viewc compila especificaciones de VIEW, mientras que viewc32 compila especificaciones de VIEW32.

Para poder utilizar este tipo de buffer se necesita seguir un conjunto de pasos que se indican a continuación:

- Definir el VIEW que corresponde a un archivo de texto con un formato específico conteniendo los campos que se desean manejar en los mensajes.
- Ejecutar viewc o viewc32 para crear una versión binaria del archivo del paso anterior y el archivo de cabecera *.h que contiene la definición de la estructura en C. Los archivos binarios quedan con el mismo nombre de los archivos de texto de entrada pero con extensión *.V
- Use #include para incluir los archivos de cabecera en los Programas Clientes y Programas Servidores que lo requieran.
- Defina las variables de ambiente que se especifican a continuación:
  Para VIEW32

Nombre Variable	**Significado**
VIEWFILES32	Es una lista separada por comas con los nombres de archivos binarios con extensión *.V.
VIEWDIR32	Es el directorio donde se encuentran los archivos binarios *.V

3.3.3.1 Especificación del Archivo de texto VIEW (viewfile)

Cada archivo de tipo view consta de tres partes:

- Una línea que comienza con la palabra VIEW (nunca con un sufijo 32), seguida por el nombre de la vista; el nombre puede tener un máximo de 33 caracteres.
- Una lista de fields o campos.
- Una línea comenzando con la palabra END.

Ejemplo de un archivo de especificación viewfile para la vista de nombre ViewPersona:

```
VIEW  ViewPersona
# type     cname          fbname        count   flag   size   null
string     Fld_Rut        FldRut          1       -      12     ""
string     Fld_Nombre     FldNombre       1       -      25     ""
```