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

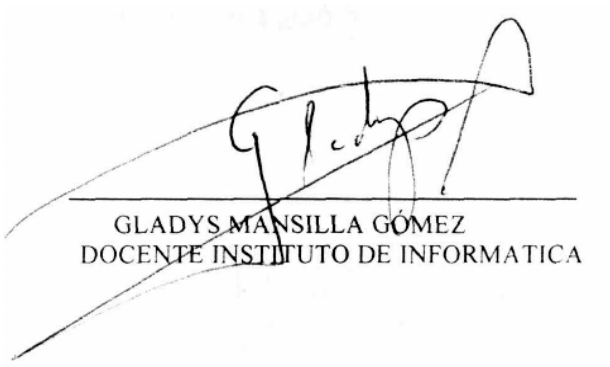
Nombre Trabajo de Titulación: "DESARROLLO DE UNA METODOLOGIA DE INTEGRACION DE SISTEMAS BASADO EN EL MONITOR TRANSACCIONAL BEA TUXEDO"

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

ÍNDICE DE FIGURAS

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.

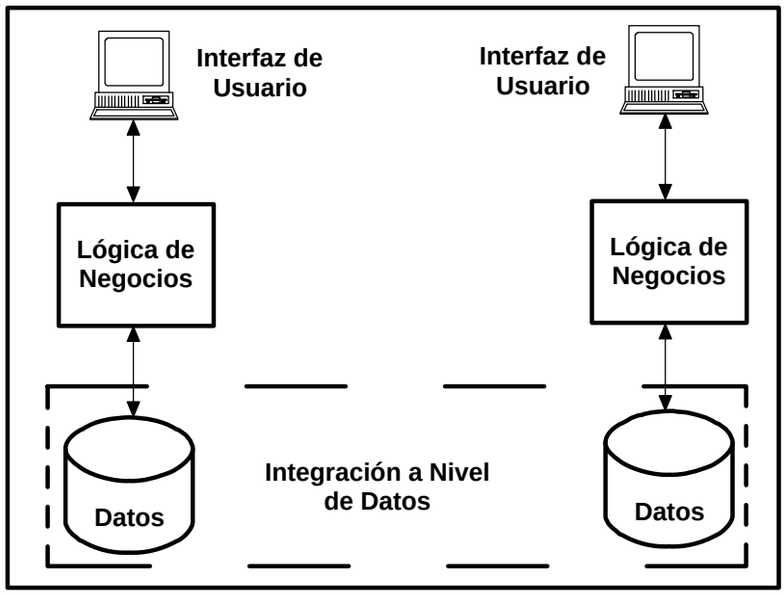


Figura 1.2-1. Integración a Nivel de Datos.

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.

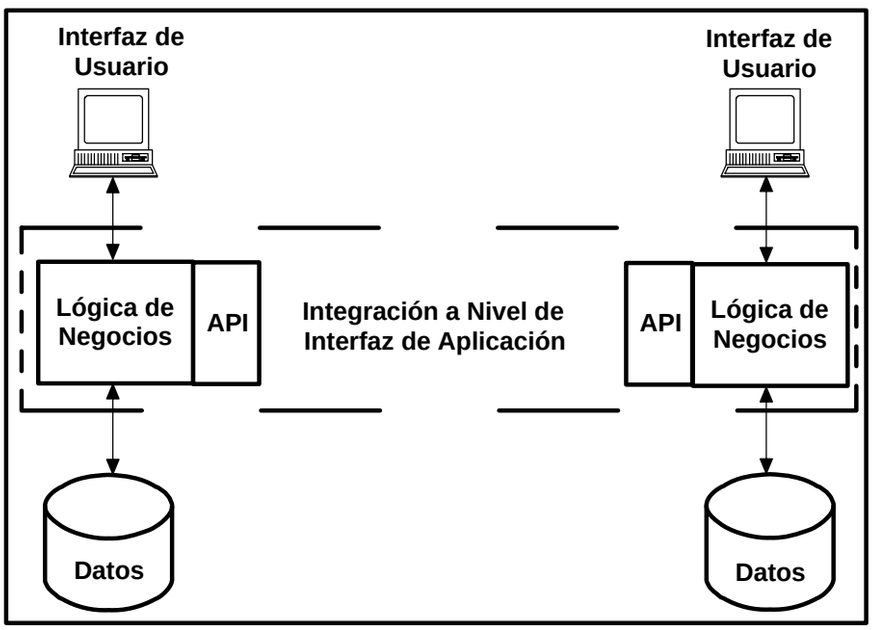


Figura 1.2-2. Integración a Nivel de Interfaz de Aplicación.

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.

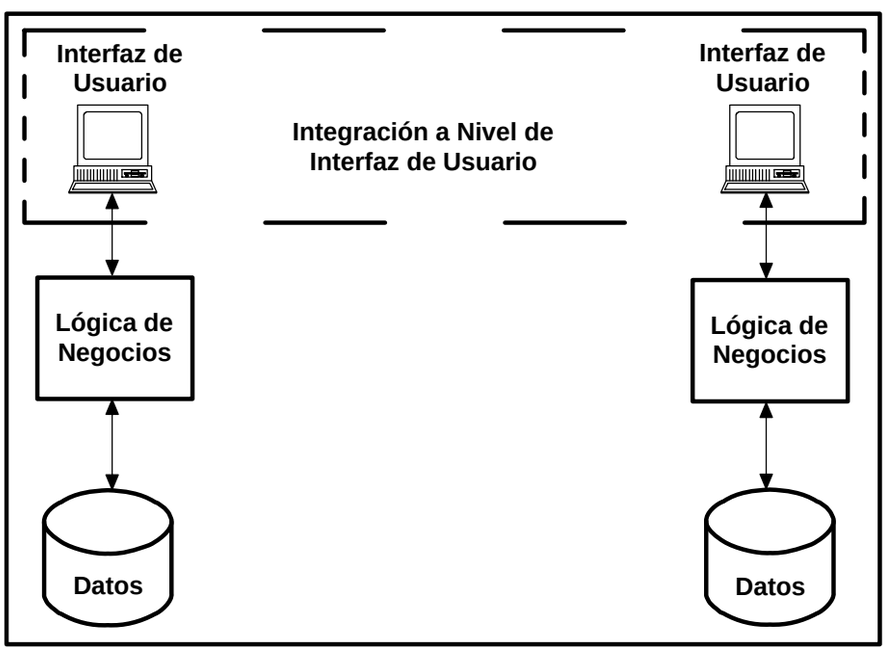


Figura 1.2-3. Integración a Nivel de Interfaz de Usuario.

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.

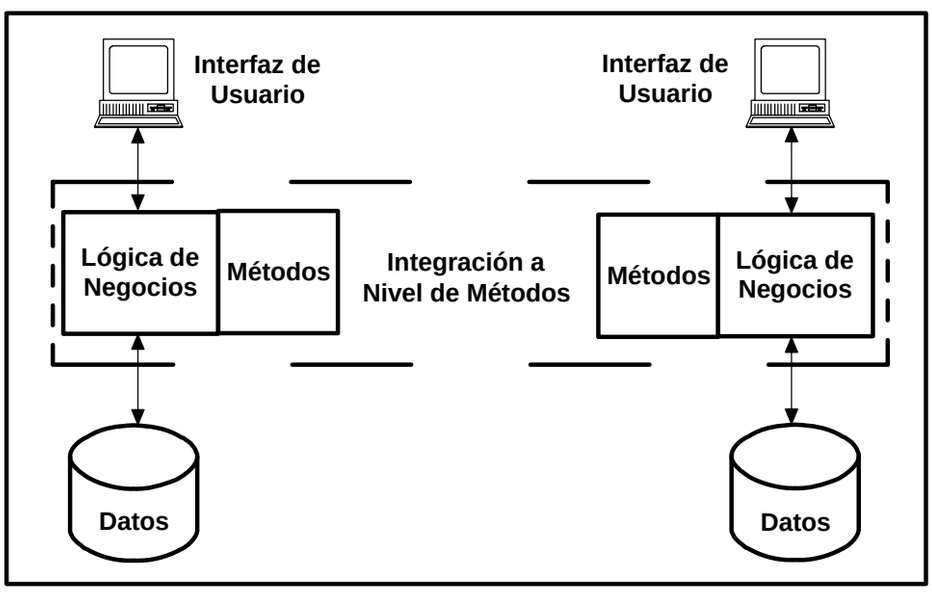


Figura 1.2-4. Integración a Nivel de Métodos.

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).

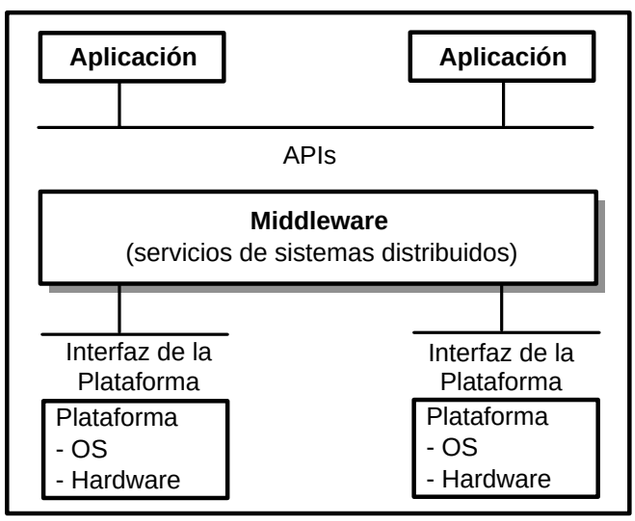


Figura 2.1-1: Concepto de Middleware.

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.

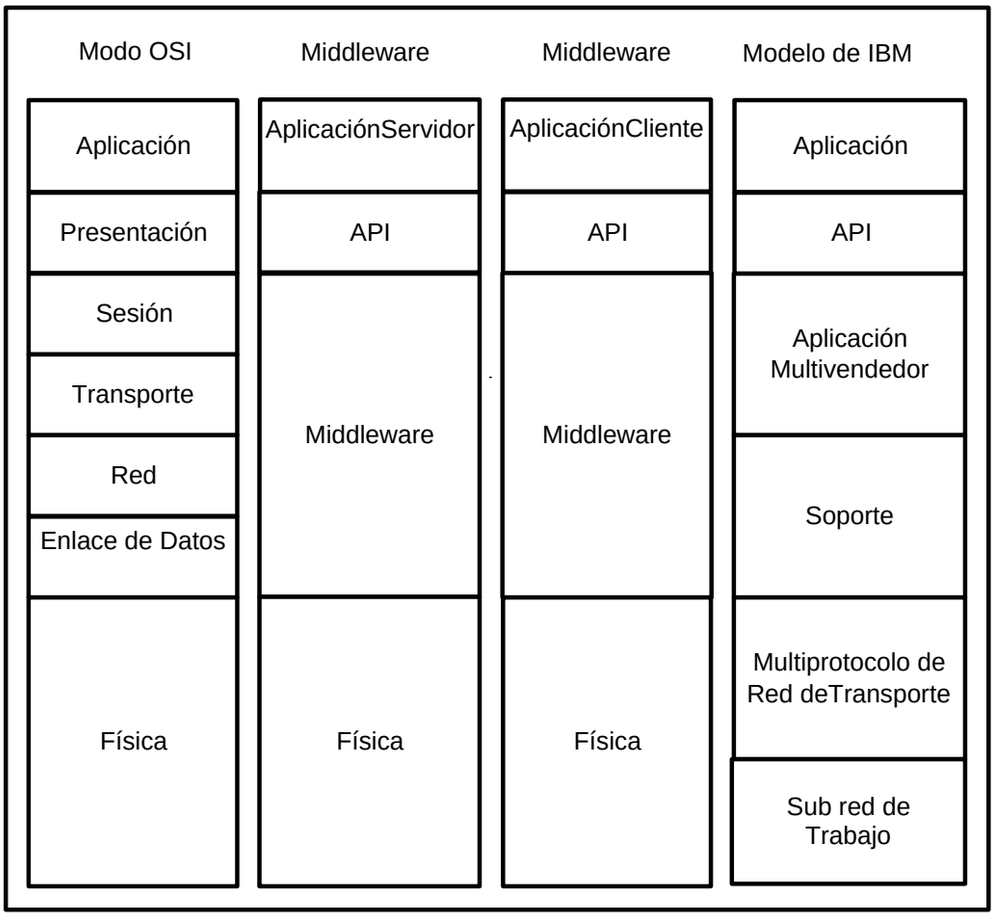


Figura 2.1-2. Concepto de Middleware en relación a un Modelo de Referencia OSI y a un Modelo de Comunicaciones de IBM.

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.

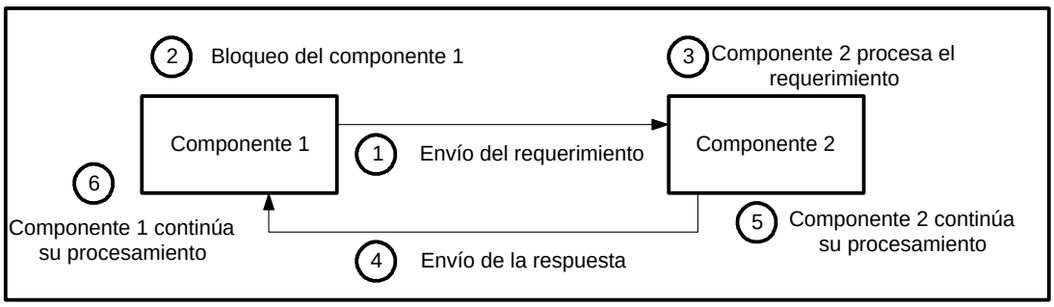


Figura 2.2-1. Comunicación Sincrónica en Requerimiento/Respuesta.

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.

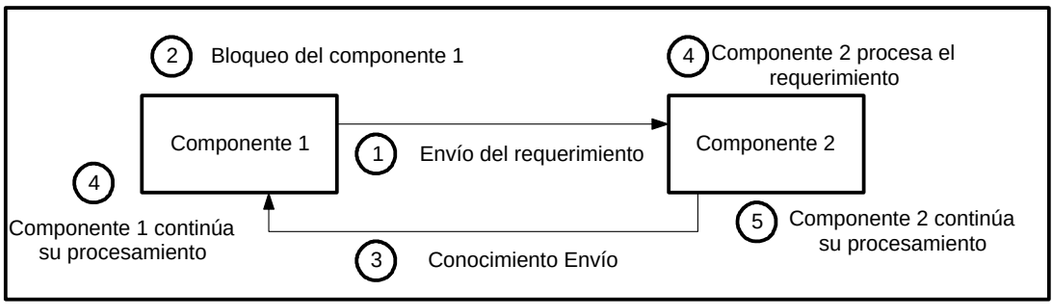


Figura 2.2-2. Comunicación Sincrónica Unidireccional.

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.

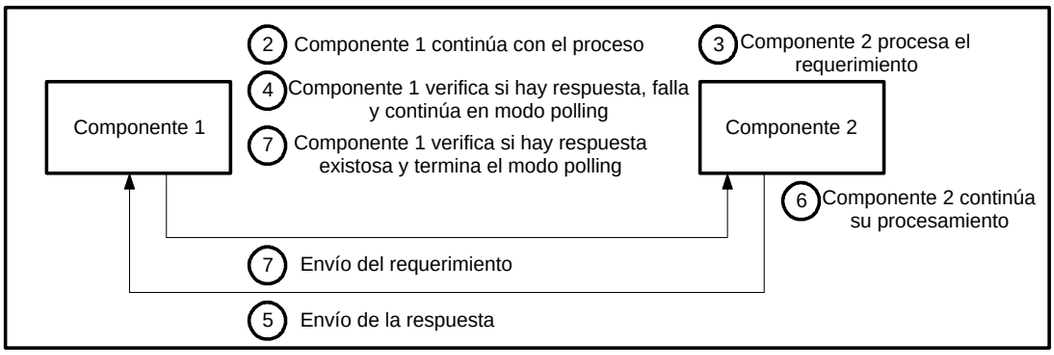


Figura 2.2-3. Comunicación Sincrónica de Polling.

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:

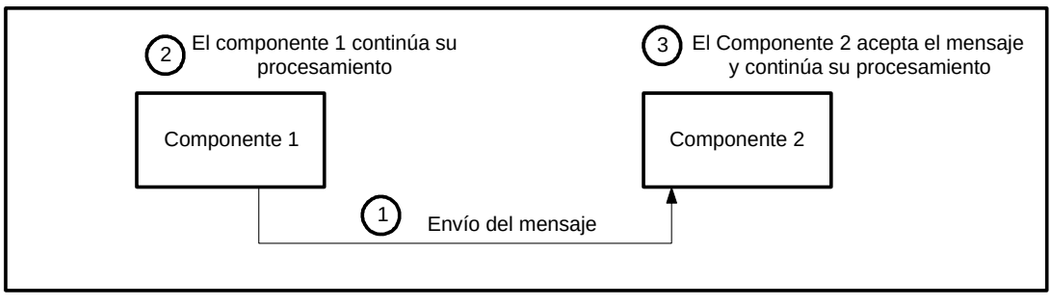


Figura 2.2-4. Intercambio de mensajes asincrónicos.

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.

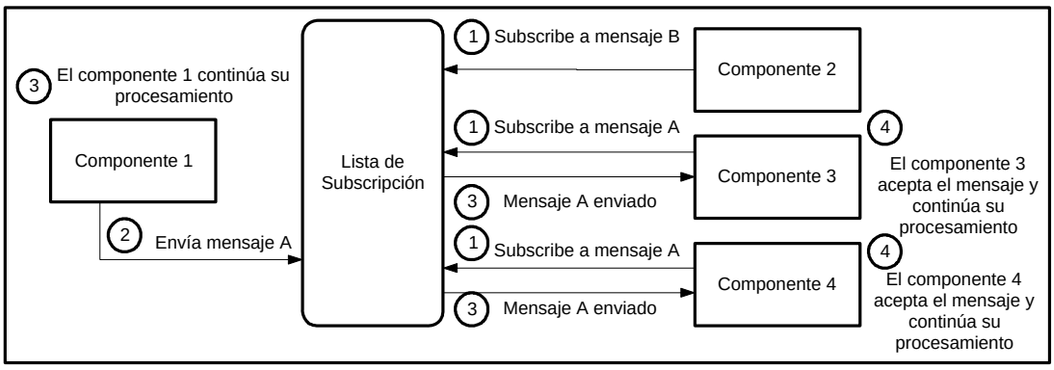


Figura 2.2-5. Mecanismo de comunicación asincrónica Publicar/Subscribir.

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.

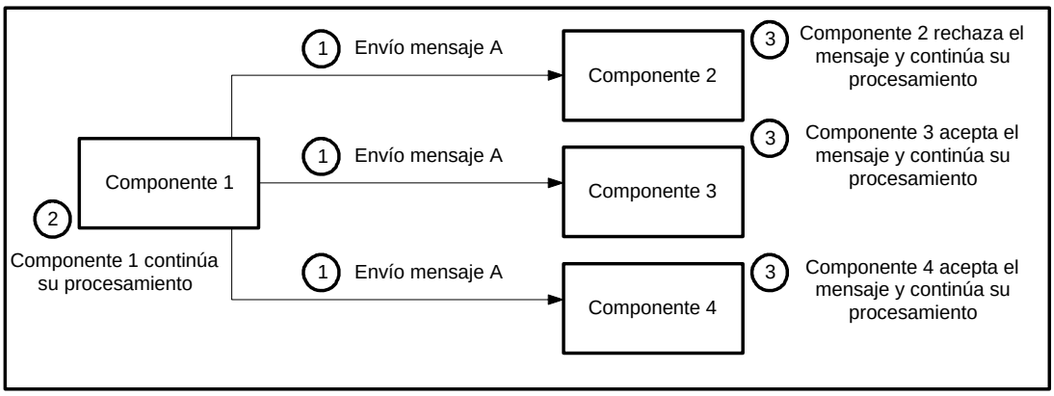


Figura 2.2-6. Modelo broadcast de comunicación asincrónica.

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.

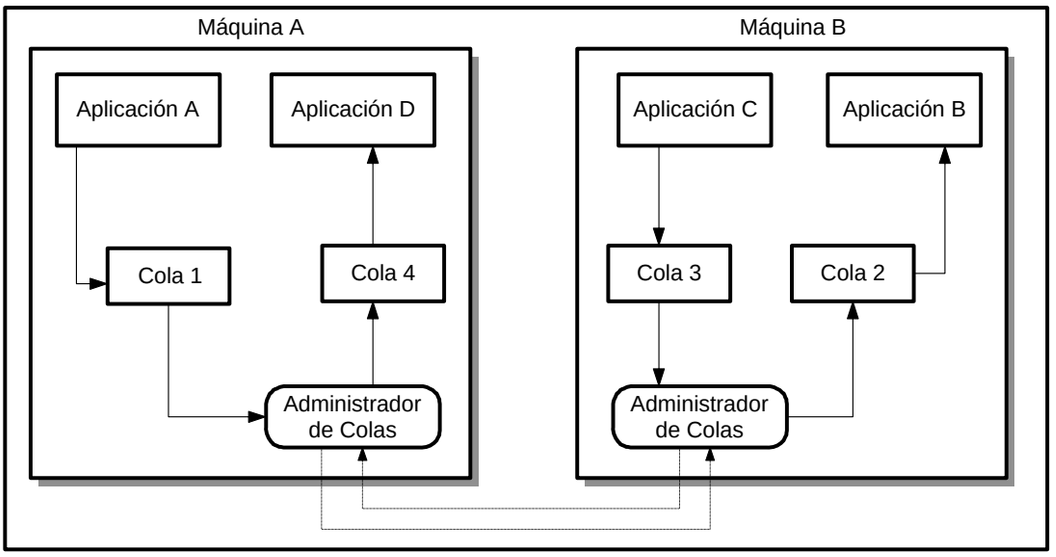


Figura 2.3-1. Modelo de Middleware Orientado a Mensajes(MOM).

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.

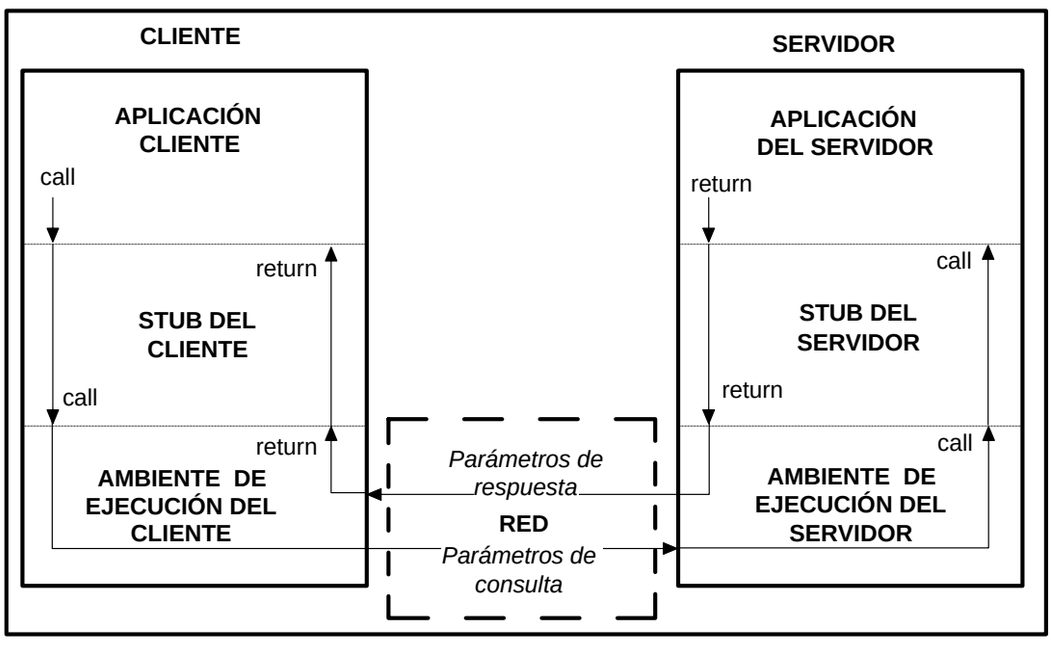


Figura 2.3-2: Modelo de Middleware RPC.

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.

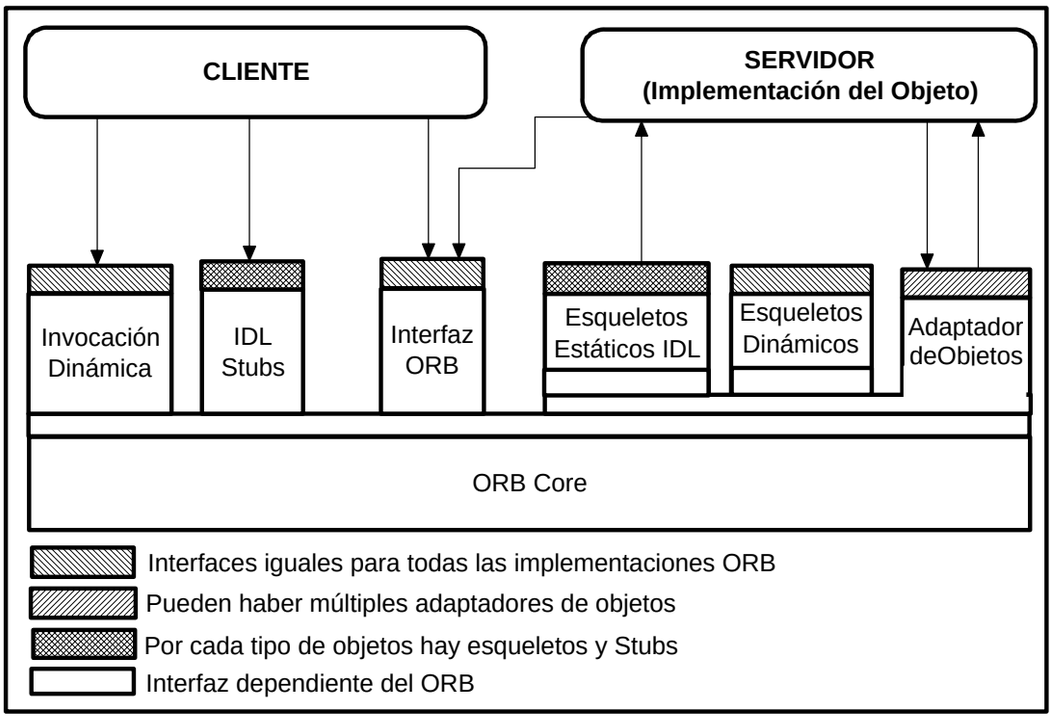


Figura 2.3-3: Estructura del ORB de CORBA

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

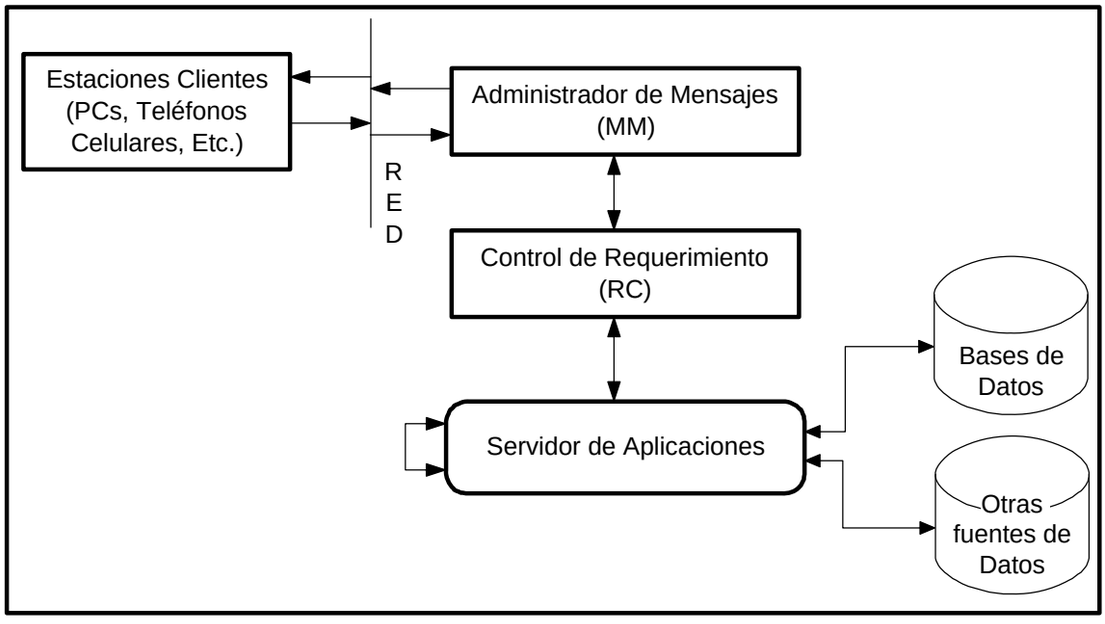
La función principal de un monitor TP es la de coordinar el flujo de transacciones requeridas entre estaciones de trabajo u otros dispositivos y programas de aplicación que puedan procesar dichos requerimientos [11]. Para lograr lo anterior, el monitor TP impone cierta estructura sobre los componentes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

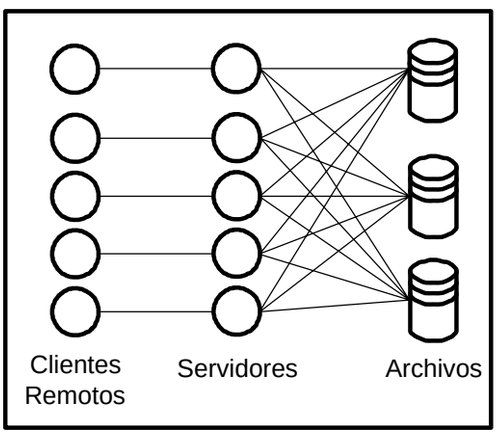


Figura 2.3-5. Modelo de proceso por cliente.

- 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.

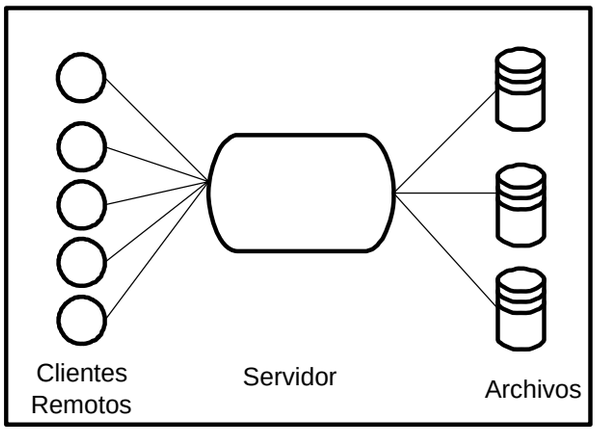


Figura 2.3-6. Modelo de único proceso.

- 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.

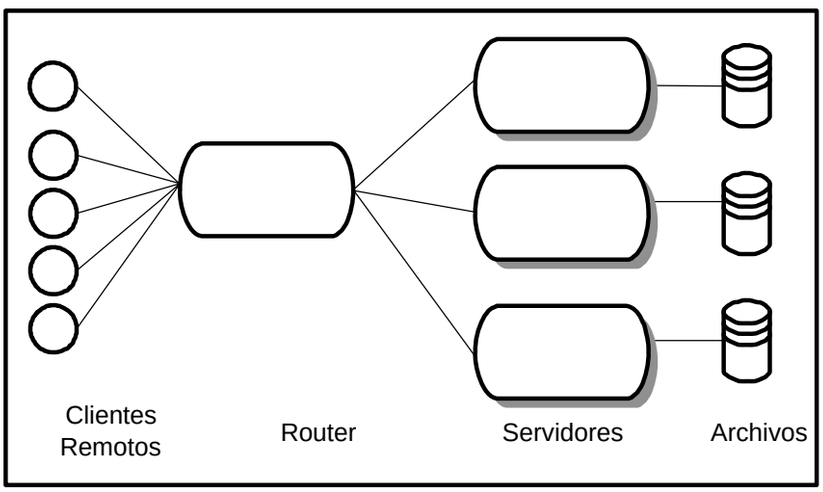


Figura 2.3-7. Modelo de muchos servidores y un único router.

- 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.

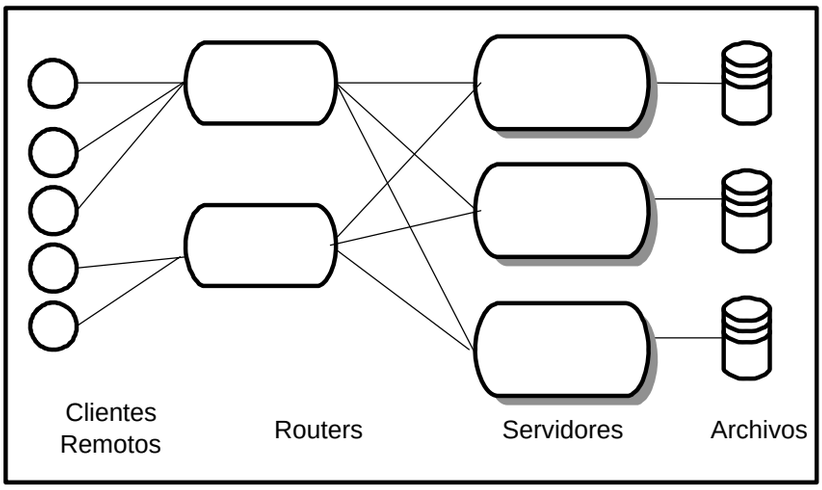


Figura 2.3-8. Modelo de muchos servidores y muchos routers.

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.

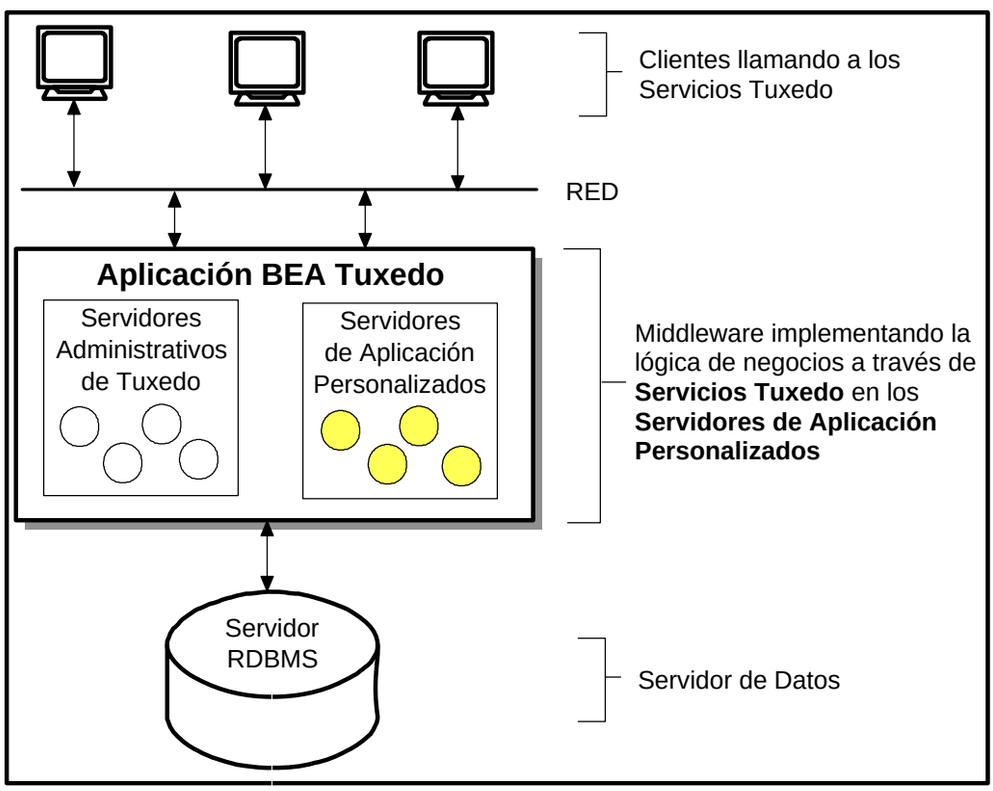


Figura 3.1-1: Concepto de BEA Tuxedo.

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.

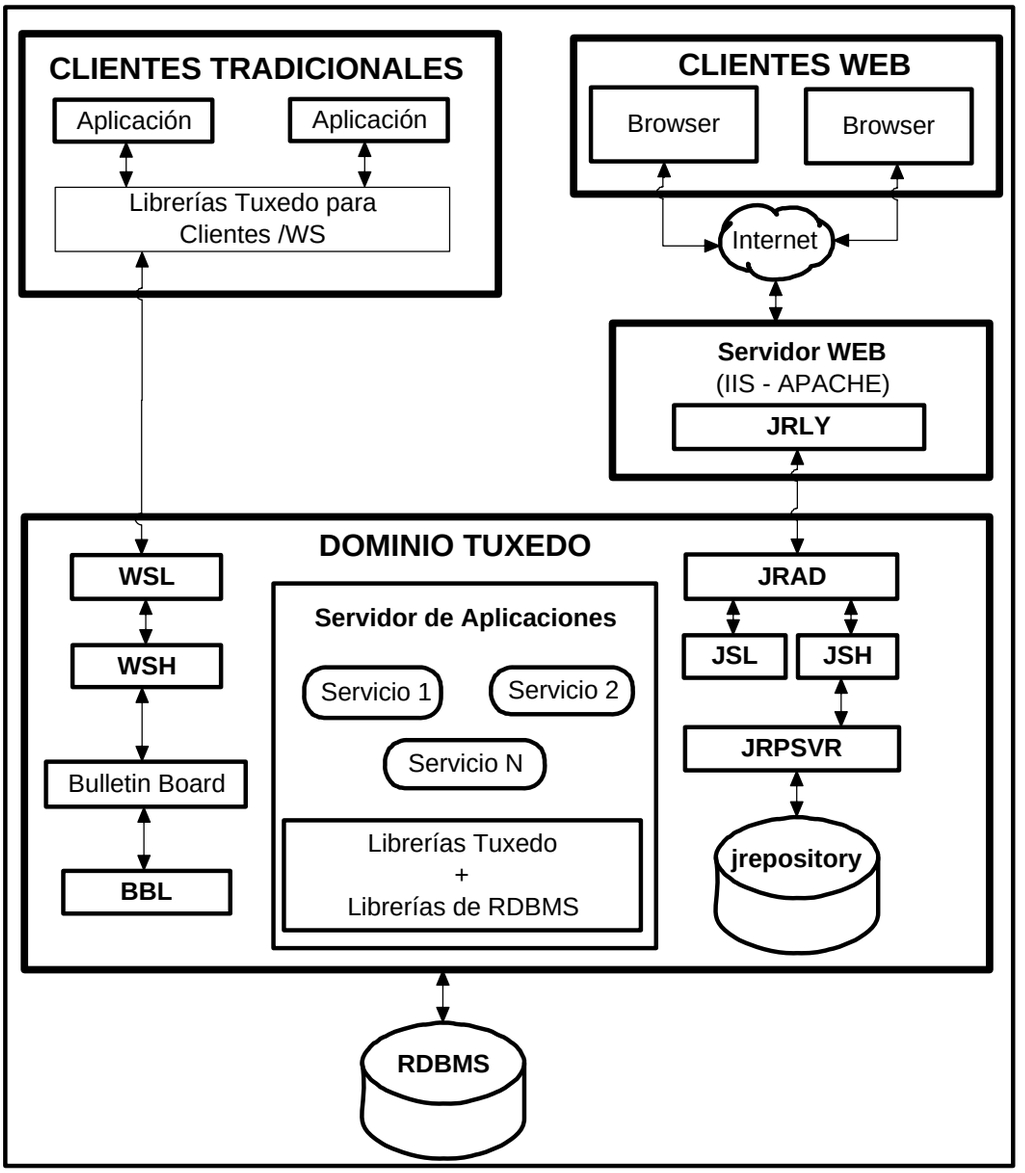


Figura 3.2-1 Elementos de una Aplicación Tuxedo.

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     ""
```

```
long    Fld_Edad    FldEdad     1    -    -    -1
string  Fld_Msg     FldMsg      10   -    100  “”
END
```

Estructura de las columnas:

COLUMNA	**SIGNIFICADO**
type	Corresponde al tipo al que pertenece el campo, los valores posibles son: char, string, carray, long, short, float, double
cname	Es el nombre del campo con el que se creará en la estructura en C o copy COBOL.
fbname	Es el nombre que existe en el archivo de definición de FML32. Puede ser omitido si no se desea que se correlacione con un campo FML32. Debe tenerse en cuenta que los nombres **cname** y **fbname** deben ser distintos. Esta opción se utiliza principalmente cuando se desea trabajar en COBOL con Buffers de tipo FML32.
count	Es la cantidad de ocurrencias del campo en la estructura en C, si se indica un valor mayor a uno, se convertirá en un arreglo en la definición de la estructura
flag	Es una lista de opciones que se pueden especificar para un campo específico.
null	Es el valor que se entenderá por nulo en la estructura y así distinguir valores válidos de los no válidos

3.3.3.2 Compilación del Archivo de texto VIEW (viewfile)

Dependiendo de la versión de VIEW o VIEW32 se deberá utilizar **viewc** o **view32** respectivamente, pero la lógica de uso y resultado son los mismos. Asumiendo que estamos utilizando VIEW32 el comando sería el siguiente:

```
viewc32  [-n]  [-d directorio_destino]  [-C] viewfile  [viewfile ...]
```

Donde viewfile es el nombre del archivo con el que fue grabada la especificación del paso anterior. directorio_destino es una opción que permite cambiar el directorio destino, si no se especifica, el resultado se dejará en el directorio actual.

La opcion [-n] se usa cuando no se necesita mapear contra un archivo FML32.
La opción [-C] permite crear un archivo COBOL COPY.
En nuestro ejemplo, si ejecutamos el comando *viewc32 ViewPersona*, se crearán como resultado de la compilación dos archivos, el archivo de cabecera para C que se llama ViewPersona.h y la representación binaria del mismo que se grabará con el nombre ViewPersona.V
En nuestro caso, el archivo de cabecera ViewPersona.h quedaría con la siguiente estructura:

```
struct ViewPersona {
   char Fld_Rut[12];        /* null="\0" */
   char Fld_Nombre[25];     /* null="\0" */
   long Fld_Edad;           /* null=-1   */
   char Fld_Msg[10][100];   /* null="\0" */
};
```

Observe que el campo Fld_Msg fue definido como una arreglo de cadenas o string.
Si ejecutamos el siguiente comando con la opción -C se genera el archivo VIEWPERSONA.cbl que se adjunta.

```
viewc  -C  ViewPersona
```

La definición Cobol que se genera es la siguiente:

```
      *      VIEWFILE: "ViewPersona"
      *      VIEWNAME: "ViewPersona"
       05    FLD-RUT                    PIC X(12).
*            NULL="\0"
       05    FLD-NOMBRE                 PIC X(25).
*            NULL="\0"
       05    FILLER PIC X(03).
       05    FLD-EDAD                   PIC S9(9) USAGE IS COMP-5.
*            NULL=-1
```

```
        05    FLD-MSG OCCURS 10 TIMES    PIC X(100).
*             NULL="\0"
```

3.3.3.3 Uso de los Archivos Generados

En runtime Tuxedo se usa la versión binaria *.V de la vista. Para que Tuxedo pueda encontrar dichos archivos se necesitan dos variables de ambiente, una conteniendo una lista de directorios donde están los archivos binarios y otra variable de ambiente que contiene los nombres de los archivos binarios *.V

3.4 CLIENTES TUXEDO

En Tuxedo existen dos tipos de clientes, Nativos y Workstations(/WS). A nivel de codificación entre uno y otro no hay diferencia alguna. Lo que los hace distintos es la forma en que se compilan, dónde pueden residir cada uno de ellos y las variables de ambiente que utilizan.

A continuación se muestra una figura que presenta ambos tipos de clientes y las variables principales que permiten conectarse a Tuxedo.

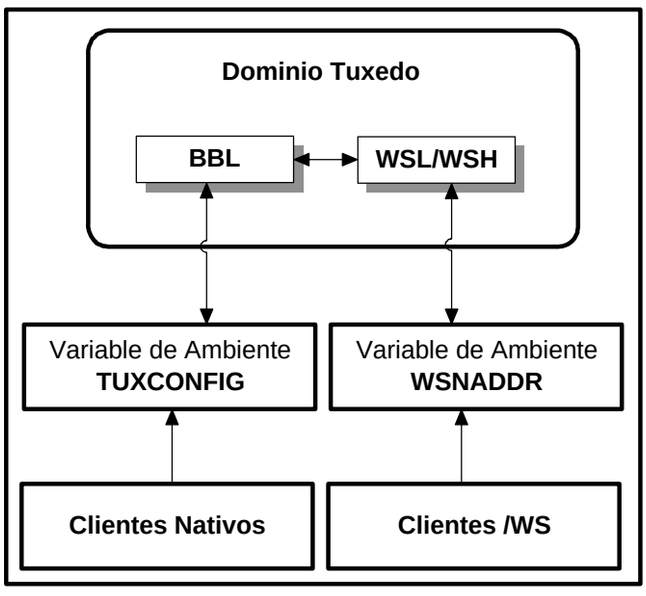


Figura 3.4-1. Tipos de Clientes Tuxedo.

3.4.1 Interfaz de Aplicación para el Control de Transacciones

BEA Tuxedo provee un conjunto de APIs conocidas como **Interfaz de Aplicación para el Control de Transacciones** (ATMI), que es una interfaz

para las comunicaciones, transacciones y administración de buffers que trabajan en todos los ambientes soportados por el sistema BEA Tuxedo. Provee la conexión entre programas de aplicación y el sistema BEA Tuxedo. Hay que indicar que estas funciones pueden ser utilizadas por los programas clientes y servidores dependiendo del contexto en el que se esté.
Dentro de las principales funciones que tienen estas APIs están:

- Verificar el nivel de seguridad de un dominio.
- Conectarse y desconectarse de un dominio.
- Administración de buffers para el intercambio de mensajes.
- Manejo de prioridad de mensajes.
- Iniciar comunicaciones del tipo Requerimiento/Respuesta.
- Iniciar comunicaciones del tipo Conversacional.
- Manejo de colas.
- Comunicación basada en eventos.
- Manejo de transacciones.
- Manejo de puntos de inicio, término y salida de servicios Tuxedo.
- Publicación dinámica de servicios Tuxedo.
- Administración de Recursos.

En general los nombres de estas APIs tienen la forma **tp*(...)**, ejemplo: tpinit(), tpcall(), tpterm().

Por último hay que señalar que los programas servidores se escriben normalmente en C o COBOL y, en el caso de C, la llamada a estas APIs es directo, pero en el caso de COBOL existe una función análoga, salvo en el caso de las APIs para el manejo de buffers que simplemente no existen.
En el caso de los clientes Tuxedo, existirán las mismas funciones ya sea en una librería compartida para sistemas operativos como UNIX, o en una DLL en sistemas operativos Windows. Por lo tanto, en este último caso, es posible escribir clientes prácticamente en cualquier ambiente de desarrollo(Visual Basic, Delphi, Centura, etc.) que sea capaz de invocar funciones en DLLs.

3.4.2 Clientes Nativos

Como ya se ha mencionado, desde un punto de vista de codificación, los clientes nativos y clientes /WS no presentan diferencia alguna. Sin embargo, los clientes nativos poseen un mecanismo de conectividad distinto al de los clientes /WS. Es así como un cliente nativo posee la capacidad de conectarse directamente al BBL de un dominio Tuxedo. La única condición es que el cliente nativo se ejecute en un nodo que tenga acceso al dominio en forma directa. Para que un cliente nativo se conecte a Tuxedo debe utilizarse la variable de ambiente **TUXCONFIG** que contiene la ubicación y nombre de la versión binaria del archivo de configuración del dominio. Ejemplos de clientes nativos son los programas de Tuxedo tmadmin y ud32.
Al momento de compilar un cliente nativo se utiliza el comando buildclient sin la opcion -w.

buildclient -v -o <ejecutable> -f <first files> -l <last files> -C

Donde:
-v, modo verbose.
-o <ejecutable>, corresponde al nombre del ejecutable o cliente final.
-f <first files>, son archivos a ser incluidos antes de incorporar las librerías de Tuxedo, de haber más de uno se encierra la lista completa en comillas dobles separados por espacios.
-l <last files>, son archivos que se agregan despues de las librerías de Tuxedo.
-C, invoca al compilador de COBOL en lugar de un compilador de C.

3.4.3 Clientes /WS

Los clientes Workstation, en adelante clientes /WS, son clientes que pueden ser ejecutados en cualquier computador o nodo de la red, con el sistema operativo que el usuario desee. Para lograr que un cliente /WS se conecte a un Dominio Tuxedo lo debe hacer a través de una variable de ambiente llamada WSNADDR que especifica la IP y Puerta por la que escuchan los procesos servidores. Por otra parte, en el Dominio Tuxedo existirá uno o más procesos llamados WSL que estarán escuchando en la IP y Puertas

antes mencionadas. La figura 3.4-2 muestra estos elementos y la manera como se relacionan:

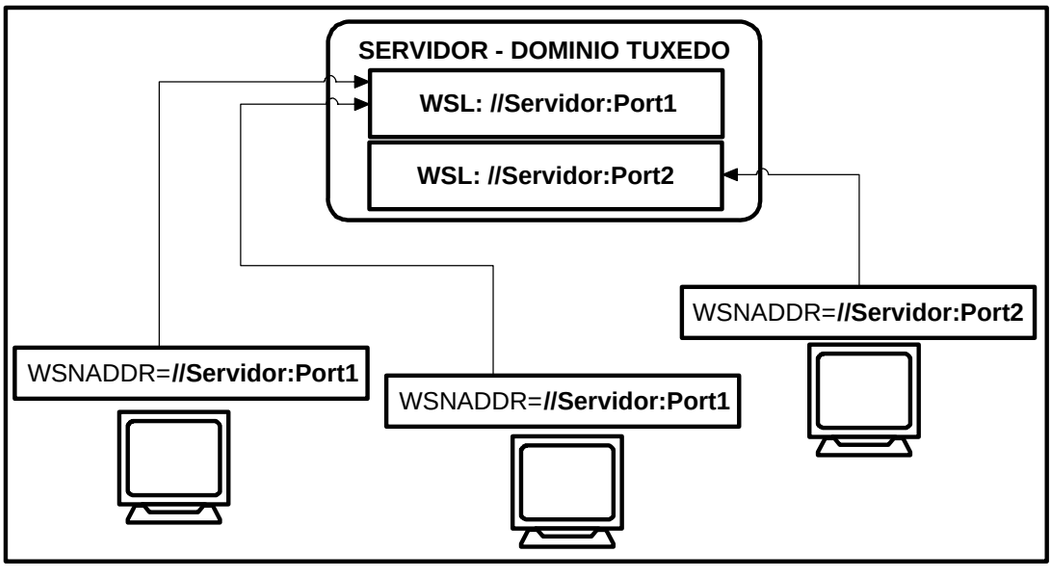


Figura 3.4-2. Cliente /WS.

Los procesos del Servidor WSL y sus respectivas puertas se especifican en el archivo de configuración del Dominio Tuxedo *.ubb como se muestra en el siguiente ejemplo:

***SERVERS**

...

...

WSL SRVGRP=WS
SRVID=10
SEQUENCE=950
GRACE=0
MAX=1
CLOPT="-A -- -n //llico:41000 -m 1 -M 16 -x 50 -T 90"

3.4.4 Estructura de Clientes

En esta sección se verá la forma en que se estructuran los programas clientes Tuxedo.

Esencialmente un cliente debe seguir los siguientes pasos:

- Conectarse a un dominio Tuxedo.

- Alocatear buffers para los parámetros del servicio al que se llamará.
- Agregar los parámetros al buffer.
- Llamar al servicio Tuxedo.
- Procesar datos recibidos del servicio.
- Liberar los buffers utilizados en la llamada al servicio.
- Desconectarse del dominio Tuxedo.

La figura 3.4-3 muestra en términos gráficos los pasos que cualquier cliente Tuxedo debiese implementar.

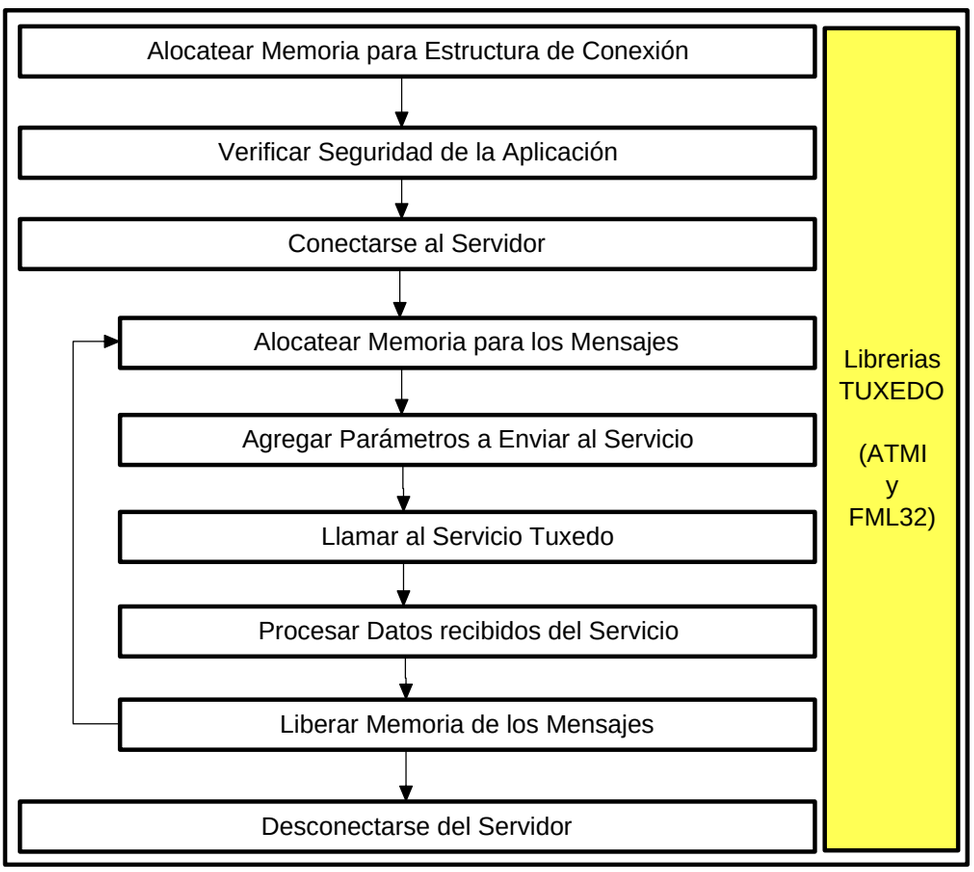


Figura 3.4-3. Estructura de un Cliente Tuxedo.

Estos pasos los debe seguir cualquier cliente Tuxedo, independiente del lenguaje en el que se programe, la diferencia estará en que varios de esos pasos pueden estar encapsulados en tan solo un método, como podría ser el caso de un lenguaje orientado a objeto, sin embargo, ese método necesariamente hará todos los pasos en forma transparente para el desarrollador.

En términos simples los clientes deben utilizar las APIs ATMI que son las responsables de la conexión y desconexión de Tuxedo, alocateo de

memoria para los buffers, llamada a servicios y chequeo de seguridad y en el caso que se esté trabajando con mensajes FML32 se deberá utilizar las APIs correspondientes. En este caso, las APIs del tipo FML32 son las que nos permiten agregar fields a los Buffers FML32, modificar fields, obtener datos, eliminar, verificar la presencia de ciertos campos, en resumen, son las encargadas del manejo de los datos que van y vienen del servidor Tuxedo.

3.5 SERVIDORES DE APLICACIÓN TUXEDO

Los **Servidores de Aplicación Tuxedo** son los programas que procesan los requerimientos de los clientes. La lógica de negocios que contienen los servidores está encapsulada en procedimientos o funciones de procesamiento denominados **Servicios Tuxedo**. Un servidor de aplicaciones Tuxedo o simplemente servidor Tuxedo puede contener uno o más servicios Tuxedo. En estricto rigor, es el servicio quien resuelve el requerimiento del cliente. Por otro lado, es interesante notar que cuando un cliente invoca un servicio específico, este no sabe cual es el servidor que contiene el servicio solicitado. Por último, se debe indicar que es responsabilidad del diseñador de aplicaciones definir la manera en que se agruparán los servicios en los servidores Tuxedo.

3.5.1 Servicios Tuxedo

Las **Propiedades de un Servicio Tuxedo** son:

- Un servicio es una rutina en C o procedimiento COBOL que implementa una tarea de negocios.
- Una rutina de servicio tiene un nombre lógico, por ejemplo “svcGetCliente”, que es utilizado por los clientes al momento de hacer un requerimiento. Este nombre puede ser igual al nombre de la rutina, o puede ser un alias para el. Así, una aplicación debería mapear el nombre del servicio “svcGetCliente” a una rutina en C con nombre svcgetcliente.
- Una rutina de un servicio tiene parámetros de entrada y salida bien definidos tal que los clientes puedan usarlos correctamente. Por ejemplo, el servicio “svcGetCliente” podría tener al menos como parámetro de entrada el rut del cliente y como parámetro de salida su nombre y dirección.

Desde el punto de vista del cliente, esas tres propiedades definen un contrato para una rutina de un servicio Tuxedo. El contrato está compuesto de:

- El nombre del servicio.
- Los parámetros de entrada que necesita el servicio.
- El procesamiento específico sobre los parámetros de entrada.
- Los parámetros de salida o lista de errores que serán devueltos al cliente como resultado del procesamiento.

La importancia de estos contratos es que una vez que ellos han sido definidos, el desarrollador del cliente y el desarrollador del servicio pueden implementar sus respectivos programas completamente independiente el uno del otro [12].

3.5.2 Estructura de Servidores

Como se mencionó anteriormente un Servidor de Aplicaciones Tuxedo puede contener uno o más servicios, pero adicionalmente podría contener al menos dos rutinas que indican el inicio de la ejecución del servidor y el término de la ejecución del servidor.

Las rutinas en cuestión son:

- tpsvrinit(), esta rutina la llama el ambiente de ejecución de Tuxedo en el instante que el servidor es ejecutado. Esta rutina por ejemplo, debiese contener el código de conexión a una base de datos, la cual sería permanente en el transcurso de vida del servidor. En el caso que el desarrollador no provea una implementación para esta rutina el sistema proveerá una por defecto.
- tpsvrdone(), esta rutina la llama el ambiente de ejecución de Tuxedo en el instante que el servidor es detenido. Esta rutina podría contener por ejemplo la lógica asociada a la desconexión de la base de datos. En el caso que el desarrollador no provea una implementación para esta rutina el sistema proveerá una por defecto.

La figura 3.5-1 muestra las rutinas descritas anteriormente.

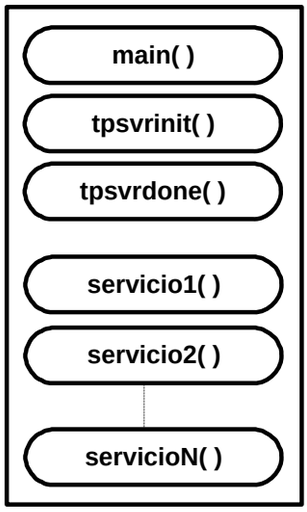


Figura 3.5-1. Estructura de un Servidor Tuxedo.

Debe observarse en la figura que también hay una rutina main() que es generada dinámicamente por el sistema Tuxedo cada vez que se compila el servidor, en consecuencia no hay forma de modificar dicha rutina.

En el contexto de un dominio Tuxedo en estado normal, los servidores de aplicación son ejecutados al activar un dominio, en tal caso, el servidor seguirá la siguiente secuencia de pasos (no necesariamente en el orden indicado):

- Llamada a la función tpsvrinit(), que ejecutará su propia lógica.
- Autoregistrarse con el BBL como disponible para recibir requerimientos.
- Esperar por la disponibilidad de mensajes (requerimientos).

3.6 SEGURIDAD EN TUXEDO

3.6.1 Seguridad a Nivel de Servidor

La seguridad de las aplicaciones Tuxedo es una característica que depende de la estrategia de seguridad de la empresa. El administrador de Tuxedo (responsable de los archivos de configuración) es quien especifica para una aplicación el nivel de seguridad requerido por ésta. Sin embargo, mencionaremos que Tuxedo maneja los siguientes niveles de seguridad a nivel de configuración:

- Sistema Operativo (NONE): En este nivel, la seguridad está delegada en las capacidades que provea el sistema operativo. Sin embrago, bajo este esquema, todo intento de conexión a una aplicación Tuxedo con la API

"tpconnect", resultará exitoso, y por lo tanto, acceso garantizado a todos los recursos que administre Tuxedo.

- Clave de Aplicacion (APP_PW): En este nivel se exige una palabra clave por aplicación Tuxedo para poder conectarse. Si el cliente entrega la clave correcta, el cliente tiene acceso a todos los recursos administrados por la aplicación Tuxedo.
- Validación de Usuario (USER_AUTH): En este nivel, la aplicación cliente debe entregar en el intento de conexión un nombre de usuario y clave válidos para tener acceso a la aplicación Tuxedo. Para implantar este esquema de seguridad, Tuxedo administra una base de usuarios, en donde están registrados todos los usuarios autorizados a usar la aplicación, con su respectiva clave. La estructura de estos archivos es similar a la usada por el sistema operativo Unix.
- Listas de Control de Acceso Opcionales (ACL): Las listas de control de acceso además de realizar la autenticación del usuario, permiten verificar si el usuario tiene acceso a algún recurso del sistema. Este chequeo se realiza cada vez que el usuario haga uso de un recurso. De no existir una ACL asociada con el usuario, Tuxedo asume que tiene permiso para acceder a todos los recursos.
- Listas de Control de Acceso Obligatorias (MANDATORY_ACL): El segundo nivel de seguridad ACL es MANDATORY_ACL, el cual exige que exista una ACL para cada recurso. De no existir una ACL, el usuario no podrá acceder al recurso.

Los niveles de seguridad se implementan en el archivo de configuración de Tuxedo (*.UBB). Tuxedo tiene su propio servidor denominado **AUTHSVR** que contiene el servicio **AUTHSVC** que implementa la seguridad, esto no significa que la empresa pueda implementar sus propios servicios de seguridad en el servidor mencionado anteriormente.

A continuación se muestra un ejemplo del archivo de configuración que implementa el nivel de seguridad USER_AUTH.

***RESOURCES**

SECURITY "USER_AUTH"

AUTHSVC "AUTHSVC"

***GROUPS**

gAUTHSVR LMID=NODO1
GRPNO=20
OPENINFO=NONE

***SERVERS**

AUTHSVR SRVGRP=gAUTHSVR SRVID=1 RESTART=Y GRACE=0 MAXGEN=2 CLOPT="-A"

Como se puede observar, las tres secciones involucradas en la configuración de la seguridad son: Recursos, Grupos y Servidores.
Por último, se debe mencionar que al especificar algún nivel de seguridad que requiera Username y Password, el cliente está obligado a proporcionar dichos datos. Sin embargo, al deshabilitar el esquema de seguridad en el servidor, no involucra necesariamente cambios en los clientes.

3.6.2 Seguridad a Nivel de Cliente

Una vez que se ha configurado el dominio Tuxedo para operar con un determinado nivel de seguridad, será necesario que cada cliente Tuxedo verifique el nivel de seguridad antes de conectarse. Para ello utiliza la API tpchkauth() y según el valor retornado se deberán actualizar el o los campos de la estructura de conexión (TPINIT). El siguiente código de ejemplo muestra la actualización de los valores:

```
if ((auth_level = tpchkauth()) == -1)
{
   userlog("Error en llamada a tpchkauth Error: %s",tpstrerror(tperrno));
   .....
   exit(1);
}
switch(auth_level)
{
   case TPNOAUTH:
      strcpy(tpinfop->cltname, AppCliente);
      break;
```

```
        case TPSYSAUTH:
            strcpy(tpinfop->cltname, AppCliente);
            strcpy(tpinfop->passwd , AppPassword);
            break

        case TPAPPAUTH:
            strcpy(tpinfop->cltname, AppCliente);
            strcpy(tpinfop->passwd , AppPassword);
            strcpy(tpinfop->usrname, Username);
            strcpy((char *)&tpinfop->data, UsernamePassword);
            break;
    }
```

Donde:

- AppCliente, es el nombre de la aplicación cliente.
- AppPassword, es la password asociada a la aplicación, normalmente se sugiere que esta variable contenga el valor "" , observe que es un par de comillas dobles sin blancos entre ellas. Esta password es la que está asociada con el dominio al momento de compilarlo con tmloadcf.
- Username, es el usuario que se conecta a la aplicación (jperez,mjimenez,etc.).
- Password, es la clave secreta asociada al Username.

En estricto rigor no es necesario actualizar todas las veces el campo cltname, pero se recomienda hacerlo para poder identificar el nombre de la aplicación cliente en el caso que existan problemas.

3.7 UTILITARIOS WUD32/UD32 DE TUXEDO

Los programas **wud32** y **ud32** son propios de toda instalación BEA Tuxedo. Estos programas son clientes Tuxedo que envían requerimientos en formato FML32. El programa ud32 es un Cliente Nativo, mientras que wud32 es un Cliente /WS.

Estos clientes reciben como entrada un archivo de texto que indica el servicio que se desea llamar y la lista de Fields de entrada y salida.

La diferencia fundamental entre estos dos clientes está en las variables de ambiente utilizadas en la conexión a un dominio Tuxedo, ya que el archivo de entrada es el mismo en ambos casos

La figura 3.7-1 refleja lo indicado a través de un ejemplo donde se realiza una llamada al Servicio Tuxedo *n_Usuario*.

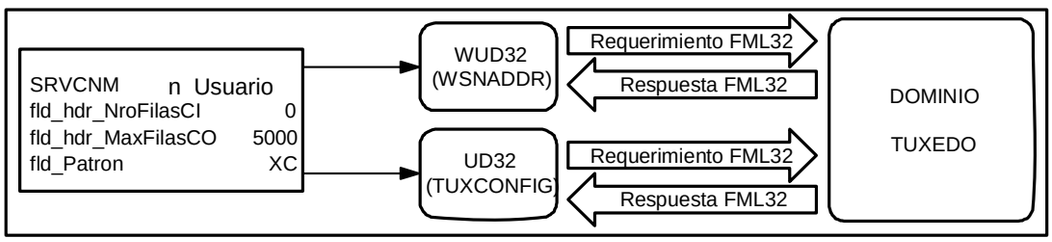


Figura 3.7-1 Utilitarios WUD32 y UD32 de Tuxedo

3.7.1 Formato del Archivo de Entrada

El archivo de entrada es igual para ambas aplicaciones y posee el siguiente formato:

SRVCNM	**<NombreServicioTuxedo>**
<NombreField1>	<ValorField1>
<NombreField2>	<ValorField2>
<NombreField3>	<ValorField3>
<NombreFieldN>	<ValorFieldN>

Donde:

Nombre Elemento	**Significado**
<NombreServicioTuxedo>	Es el nombre del servicio Tuxedo que se desea invocar.
<NombreField $_i$ >	Es el nombre del campo que espera como parámetro el Servicio Tuxedo. Aquí se especifican los parámetros de entrada, salida y entrada salida.
<ValorField $_i$ >	Es el Valor asociado al Field i.

Consideraciones:

- Los elementos <NombreField $_i$ > y <ValorField $_i$ > deben ir separados por un TAB.
- Los elementos SRVCNM y <NombreServicioTuxedo> deben ir separados por un TAB.
- El archivo debe terminar con una línea en blanco(vacía).

Errores comunes al escribir estos archivos:

- Colocar un espacio en blanco antes o después del nombre del servicio Tuxedo.
- Colocar un espacio en blanco inmediatamente después de un Nombre de Field.
- Colocar un espacio en blanco inmediatamente después de la palabra reservada *SRVCNM*.

El único elemento que puede contener uno o más blancos es el *Valor del Field i*.

3.7.2 Ambiente UD32

El ambiente para un cliente nativo ud32 es el que se indica a continuación:

Nombre Variable	**Significado**
TUXCONFIG	Contiene el **PATH** y **Nombre del archivo** binario UBB asociado a la configuración del Dominio Tuxedo.
TUXDIR	Directorio de instalación de Tuxedo.
FLDTBLDIR32	Contiene una lista separada por : directorios donde están las tablas de fields. Se recomienda incluir el directorio *udataobj* de Tuxedo.
FIELDTBLS32	Contiene una lista separada por: tablas de Fields, debe incluir la tabla Usysfl32 de Tuxedo. Ejemplo: FIELDTBLS32=Usysfl32,TblGlobal.fld
PATH	Debe contener el directorio bin donde se encuentra el programa ud32.

3.7.3 Ambiente WUD32

En la siguiente tabla se describen las variables de ambiente necesarias para ejecutar el cliente /WS wud32, que en esencia utiliza las mismas variables de ambiente del cliente nativo, salvo que se reemplaza la variable: TUXCONFIG por WSNADDR.

Nombre Variable	**Significado**
WSNADDR	Contiene la Dirección IP y Puerto donde escucha el WSL.
TUXDIR	Directorio de instalación de Tuxedo.
FLDTBLDIR32	Contiene una lista separada por: directorios donde están las tablas de fields. Se recomienda incluir el directorio *udataobj* de Tuxedo.
FIELDTBLS32	Contiene una lista separada por: tablas de Fields, debe incluir la tabla Usysfl32 de Tuxedo. FIELDTBLS32=Usysfl32,TblGlobal.fld
PATH	Debe contener el directorio /bin donde se encuentra el programa wud32.

3.8 PRUEBAS DE ESFUERZO

Las pruebas de esfuerzo son importantes porque permiten determinar cómo se comportará una determinada solución ante un nivel específico de requerimientos. Esto permitirá evitar problemas graves cuando los sistemas se pasen a explotación. A continuación se definirán un conjunto de pasos que ayudarán a definir un esquema de pruebas.

- Definir un universo de clientes simultáneos que podrían potencialmente utilizar el sistema en una instante T. Por ejemplo, si se trata de una aplicación para usuarios dentro de la empresa es relativamente fácil determinar el número de clientes concurrentes. Por otro lado, si no se tiene certeza de cuantos clientes potenciales pueden existir, se puede trabajar en base a rangos de clientes, por ejemplo: 1-100, 101-500, 501-1000, etc. Esto permitirá entender cual es umbral en donde nuestra solución funciona sin problemas.
- Definir un conjunto de servicios o funciones de negocio críticas que estarán sometidas a carga. En general los sistemas explotan una funcionalidad más que otra.
- Definir métricas para poder evaluar los resultados obtenidos.
    - Uso de CPU durante las pruebas.
    - Uso de memoria durante las pruebas.
    - Tiempo de respuesta esperado por transacción.
- Para aquellos casos donde las soluciones involucren Bases de Datos, se deberá definir un conjunto de datos que permita simular el comportamiento real que tendría en producción. No es lo mismo probar una consulta sobre 1000 registros en una Base de Datos de prueba que probar con 1.000.000 de registros en producción.
- En el caso de prueba de servicios Tuxedo, se recomienda utilizar los utilitarios UD32 y WUD32, de esta manera no es necesario construir clientes en un lenguaje específico.
- Crear scripts (Ejemplo: Shell de UNIX) que a partir de un conjunto de datos de entrada, generen en forma dinámica los archivos de entrada para los utilitarios WUD32 y UD32. Asimismo ejecutar a partir de dichos archivos la ejecución de clientes concurrentes. Si fuese necesario se puede distribuir la carga de los clientes en diversas máquinas.

- Finalmente, se deberán comparar las métricas preestablecidas con los resultados obtenidos y, si corresponde, tomar las acciones necesarias para corregir los problemas.

Hay que entender, que no siempre la corrección de un problema pasa por modificar una pieza de software, hay ocasiones que cuando se integra con hardware específico, podría darse el caso de que éste pudiese ser el cuello de botella.

3.9 ANÁLISIS DEL TIEMPO DE RESPUESTA

A continuación se describirá la manera en que se pueden medir los tiempos de respuesta de los servicios Tuxedo.

Dado un Dominio D, existe un archivo de configuración D.ubb donde se especifican todos sus componentes, entre ellos un conjunto de servidores de Aplicación S1,S2,...,Sn para los cuales se necesita medir el tiempo de respuesta.

En particular, si se desea medir el tiempo de respuesta de los servicios Tuxedo del servidor S1, habrá que modificar la configuración del archivo D.ubb, sección SERVERS como se muestra a continuación:

```
*SERVERS
......
S1      CLOPT="-r -e NombreArchivoDeErrores.err"
```

La opción -r le dice al sistema Tuxedo que genere las estadísticas de tiempo de respuesta que se crearán en el archivo de errores especificado en la opción -e.

El formato de este archivo es tal, que con un simple análisis visual no es posible deducir nada respecto de los tiempos de respuesta.

Por tal motivo, este archivo debe ser procesado por los utilitarios txrp o ztxrpt, se recomienda la utilización del segundo porque entrega información más detallada de los resultados.

Los principales datos que entrega este informe son:

- Nombre del servicio.
- Tiempo mínimo de respuesta alcanzado por el servicio.
- Tiempo máximo de respuesta alcanzado por el servicio.
- Tiempo promedio de respuesta.
- Cantidad de requerimientos que sobrepasaron el tiempo máximo.

A partir de esta información se pueden generar informes por períodos que permitan establecer mejoras respecto de los servicios más lentos.

3.10 ELEMENTOS DE LA METODOLOGÍA

Una buena metodología debe asegurar que la arquitectura IT de la empresa satisfaga las necesidades del negocio, que describa como manejar las actividades de integración, que describa como integrar los sistemas heredados, soluciones empaquetadas y sistemas distribuidos, seleccionar y estandarizar una guía tecnológica, y por último, garantizar la reutilización de la lógica de negocio.

Nuestra metodología se enfocará en los siguientes puntos:

- Desarrollar una estrategia de integración que apoye las metas de negocio de la empresa.
- Desarrollar una arquitectura de la empresa que apoye la estrategia de integración y establecer estándares y guías para proyectos individuales.
- Desarrollar aplicaciones que se adhieran a la estrategia de integración y a la arquitectura de la empresa.

Las áreas que comprende la metodología de integración son:

- Estrategia IT de la empresa.
- Arquitectura de la empresa.
- Arquitectura de la aplicación.
- Desarrollo de servicios.
- Integración y distribución de la aplicación

Estas cinco áreas se construyen lógicamente una sobre otra, pero no obedecen a una relación de manera secuencial en el tiempo. Uno de los desafíos más grandes en integración, es asegurar que la arquitectura de la empresa permanezca bajo control, mientras las tareas de las cinco áreas de actividades se realicen concurrentemente.

3.10.1 Estrategia IT de la Empresa

Esta es una de las primeras actividades para una metodología de integración. Desarrollar una estrategia IT, que haga la relación entre las necesidades del negocio y las actividades IT deben ser establecidas y respaldadas por la plana gerencial de la empresa.

Lo anterior se basa esencialmente en que los costos involucrados en una solución de integración a través de un monitor transaccional Tuxedo no son despreciables, tanto en la adquisición del producto en si, como en las posibles modificaciones de los sistemas ya existentes. Esta clase de soluciones requiere de personal especializado.

3.10.2 Arquitectura de la Empresa.

La arquitectura de la empresa identifica los componentes de los sistemas de la empresa, describe como estos componentes se relacionan entre ellos, y presenta estándares, que incluyen los principios y restricciones por los cuales los componentes pueden ser refinados en diseños más detallados. El rol de la arquitectura de la empresa es restringir los diseños de elementos de las componentes para lograr las metas de arquitecturas globales, tales como la estandarización, simplificación de interfaces, y la capacidad de acomodarse a los cambios tanto en el ambiente del negocio como en las tecnologías de apoyo.

Para que la arquitectura de una empresa tenga una base efectiva para el desarrollo de aplicaciones, hay que tener presente algunos puntos y saber como dirigirlos en forma correcta:

- Definir una política de seguridad en la empresa.
- Lograr el entendimiento de los requerimientos de alto nivel de la empresa para una adecuada funcionalidad del negocio.
- Evaluación de los diversos a ser integrados.
- Se debe definir y documentar la arquitectura de la empresa.

3.10.2.1 Desarrollo de una Política de Seguridad

El propósito de definir una política de seguridad es reducir los riesgos de los recursos de información de la empresa. La inadecuada protección de los

recursos de información podría resultar en la pérdida de estos y en la pérdida de una posición competitiva o de reputación.

El objetivo de una política de seguridad de la información de una empresa es especificar metas de protección de alto nivel requeridas para ejecutar los negocios de la empresa y estrategia IT.

Las políticas de seguridad deben ser convincentes y aceptables a todos los niveles de la compañía. Las políticas además deben ser entendibles para los usuarios del sistema quienes deben adherirse a la guía de políticas de seguridad y evitar así el intento de engaños.

Las políticas deben también identificar los roles y responsabilidades individuales de quienes traten con la información sensible de la empresa.

Se deben identificar los recursos críticos de información. Los recursos críticos no siempre son datos. La lógica del negocio incluida en las aplicaciones debe ser tan importante como los datos manejados por estas aplicaciones.

Se describirán algunos pasos requeridos para desarrollar una política de seguridad:

- Definir roles y responsabilidades. Estas pueden incluir los roles del personal técnico de IT, así como también del personal de administración y de operaciones.
- Identificar los recursos de información crítica. Estos recursos deben colocarse en categorías basadas por ejemplo en información delicada, información confidencial personal, en propiedad intelectual y datos delicados de operar.
- Identificar a *quienes* se les debería permitir el acceso a los diversos tipos de información y qué acción podrían hacer sobre ellos (por ejemplo crear, actualizar, borrar, etc.)
- Determinar la aplicabilidad de la política. Por ejemplo, si es aplicable de igual manera para un empleado como para un cliente, o para alguien que está conectado directamente a una LAN de la empresa o vía otro acceso.
- Definir los requerimientos de cumplimiento. Describir el uso de la información aceptable y no aceptable, y determinar la consecuencia del uso no aceptable.

Lograr unificar en un único sistema de privilegios a toda la compañía puede resultar muy difícil, por cuanto cada sistema tiene sus mecanismos de

seguridad específicos. Eventualmente, en el caso de Tuxedo se puede desarrollar un servidor de seguridad que se integre a algunos de los sistemas de seguridad ya existente.

3.10.2.2 Analizar los Requerimientos de la Empresa

Analizar los requerimientos a nivel de la arquitectura de la empresa tiene dos objetivos:

- Determinar la funcionalidad de alto nivel de la empresa que será compartida a través de las aplicaciones Tuxedo.
- Determinar los servicios de la infraestructura que deben estar disponibles para las aplicaciones.

Basados en los modelos de procesos, podemos definir los componentes de alto nivel del negocio.

Las componentes de alto nivel de la empresa deben ser agrupados de acuerdo a la función de negocio que los crea.

3.10.2.3 Analizar los Requerimientos de Infraestructura

El propósito de analizar los requerimientos de infraestructura a nivel de la arquitectura de una empresa es identificar las características que la arquitectura de la empresa debe tener para proveer niveles aceptables de servicios. Los requerimientos de arquitectura de la empresa no deben sólo dirigirse hacia las actuales necesidades de la empresa, sino que especificar las características de la arquitectura que deben poseer para asegurar que los sistemas IT de la empresa puedan apoyar los cambios necesarios.

Para comenzar a determinar los requerimientos de infraestructura, las aplicaciones planificadas podrían ser agrupadas en categorías basadas en las siguientes características:

Grupos de Usuarios. Clientes, empleados de socios o empresas aliadas y empleados dentro de la empresa.

Escala de Uso. Grupos de usuarios, cientos, miles, etc.

Grado de Criticidad: misión crítica, no crítica.

Con cada categoría, analizar las aplicaciones con respecto a los siguientes puntos:

- Requerimientos de seguridad.

- Mantención y adaptabilidad.
- Confiabilidad y disponibilidad.
- Desempeño.
- Escalabilidad.
- Integridad.
- Usabilidad.
- Recuperabilidad.

Usando este criterio de agrupación de aplicaciones, podemos identificar la infraestructura de servicios necesarios. Es importante ser realista en la evaluación. Cada una de estas características tiene un costo, tanto en términos de tiempo de desarrollo como en productos de infraestructura adicionales. Los requerimientos de los servicios deberían reflejar las verdaderas necesidades de las aplicaciones.

3.10.2.4 Evaluación de Aplicaciones

La evaluación de las aplicaciones heredadas y empaquetadas para determinar como encajan en el diseño completo es un aspecto crítico al definir la arquitectura de una empresa. El método en las dos categorías es esencialmente el mismo.

Integrar este tipo de aplicaciones es un trabajo muy complejo, porque en general involucra la adquisición de un adaptador que pueda exportar la funcionalidad como servicios Tuxedo.

Otra alternativa es adoptar una estrategia de migración ordenada, vía un reemplazo gradual de los sistemas heredados. Cuando la migración se haya completado, el sistema heredado será totalmente reemplazado con la nueva funcionalidad, basada en tecnología Tuxedo.

Resulta natural evaluar cuidadosamente las alternativas de integración vía un adaptador versus la estrategia de migración.

3.10.2.5 Especificación de la Arquitectura IT de la Empresa.

La arquitectura de la empresa debe ser definida y documentada con suficiente detalle, lo cual permite que el desarrollo de las aplicaciones proceda de una mejor forma. La especificación debe ser bastante detallada

para que las aplicaciones puedan ser desarrolladas en forma concurrente e independiente.

En este punto es importante definir los dominios Tuxedo existentes y la relación entre ellos. Normalmente ciertos dominios tienen funcionalidades bien definidas que deben ser de conocimiento público para evitar problemas de diversa índole.

3.10.3 Arquitectura de la Aplicación

La arquitectura a este nivel asegura que la aplicación se ajusta a los requerimientos del negocio y que provee una calidad de servicio y seguridad adecuada.

Para definir la arquitectura de la aplicación se deben considerar los siguientes elementos:

- Requerimientos de una Aplicación.
- Análisis de requerimientos de Seguridad de la Aplicación.
- Desarrollo de la arquitectura de la Aplicación.

Definir arquitecturas de aplicación para un ambiente de desarrollo de la empresa permite que los diversos proyectos ajustados a esta arquitectura sean más simples de implementar. Lo anterior se debe a que la problemática de carácter técnico sea resuelta una sola vez.

3.10.4 Desarrollo de Servicios

Un servicio transaccional es una entidad de software que provee un conjunto cohesivo de capacidades funcionales a través de una interfaz específica. Una aplicación potencialmente consiste de muchos servicios, y un servicio que forma parte del dominio del negocio o de la capa de infraestructura de la arquitectura de la empresa puede apoyar muchas aplicaciones.

3.10.5 Integración y distribución de la Aplicación.

En este punto se propone realizar pruebas pilotos para determinar la factibilidad técnica de la solución de integración planteada. Una vez evaluados los resultados de las pruebas se procede al diseño y construcción propiamente tal. Se recomienda establecer mecanismos de control a través

de un departamento o sección de Control de Calidad. Adicionalmente se deberán realizar pruebas de rendimiento para garantizar un tiempo de respuesta aceptable de los servicios Tuxedo.

4 DESARROLLO DE LA INTEGRACIÓN DE APLICACIONES

Este capítulo muestra los elementos utilizados en la integración de algunos de los sistemas de Bellsouth, cumpliendo así con parte de los objetivos propuestos en esta Tesis. Adicionalmente se plantean los principales problemas con los que se enfrentaba la compañía en sus procesos de negocios, los que de una u otra forma tenían incidencia en el servicio que recibía finalmente el cliente.

4.1 SITUACIÓN ACTUAL

Bellsouth es una compañía líder en comunicaciones móviles en Chile, ofreciendo a los clientes un servicio de excelencia y un amplio rango de productos, desde comunicación de voz, larga distancia, data e Internet móvil. Para ofrecer sus servicios, Bellsouth posee una fuerza de ventas constituida tanto por oficinas externas que hacen las veces de agentes de ventas, como por sus propias oficinas. Los agentes de ventas están distribuidos a lo largo del país y utilizan los sistemas necesarios para atender a los clientes.

Los productos de Bellsouth están clasificados en dos grandes grupos:

- Productos para Personas, orientado principalmente a personas naturales.
- Productos para Empresas, orientado a compañías que requieran de servicios móviles.

Dentro de esos grupos existen obviamente los teléfonos móviles, planes mensuales y servicios al cliente, donde quizá el servicio más conocido en la actualidad sea el servicio de *Club Bellsouth*, que dicho sea de paso, está basado en la utilización de servicios Tuxedo.

4.2 PRINCIPALES PROBLEMAS

Antes que se llevara adelante el plan de integración de la compañía a través de Tuxedo, existían varios problemas, entre los principales:

- Cualquier intento de integración con SAP era difícil.
- Existen aplicaciones de proveedores externos que accedían a las bases de datos de SAP para poder implementar funcionalidad específica. Esto naturalmente es una mala solución por cuanto los productos empaquetados por naturaleza son cajas negras que no debiesen alterarse bajo ningún punto de vista. SAP provee una interfaz de aplicación para poder acceder a la funcionalidad deseada.
- Respecto de SAP, hay que mencionar que no existe documentación en la compañía relacionada a las APIs que proveen la lógica de negocios.
- Para determinados procesos de negocios, se utiliza más de una aplicación cliente, lo cual claramente degrada el rendimiento operacional. Peor aun, si se ve involucrado el cliente.
- Además de SAP, Bellsouth cuenta con aplicaciones hechas en casa bajo un modelo cliente/servidor de 2 capas, con la lógica de negocio en el lado del cliente, lo cual hace muy difícil poder reutilizar dichos procesos de negocio desde otras aplicaciones.
- No existía documentación que describiera las principales funciones de negocio, por lo tanto, para poder hacer cualquier intento de integración, se necesita recurrir a los autores de dicha funcionalidad, al personal externo que lo implementó o, en el peor de los casos, al código fuente.
- Existían procesos de servicios a clientes que eran dependientes de empresas externas que no estaban en línea, lo cual generaba procesos manuales todos los días para poder actualizar los datos en las bases de datos centrales. Lo anterior, dejaba abierta la posibilidad que un cliente de Bellsouth realizara dos veces el mismo proceso en lugares distintos durante un día.

4.3 EVALUACIÓN DE BEA TUXEDO

Si bien Bellsouth reconocía los problemas que la afectaban, definir un producto de integración para la compañía no es un tema menor. Por lo tanto,

definió los siguientes pasos para evaluar Tuxedo como una posible herramienta que sirviera para integrar sus sistemas.

- Involucrar a los principales miembros de sus áreas de desarrollo.
- Establecer un período en el que se dictaría un taller en las oficinas de Bellsouth para poder evaluar y comprender las características de BEA Tuxedo.
- Definir las herramientas básicas de desarrollo de la compañía, en base a las cuales se harían las pruebas. En tal sentido, las herramientas utilizadas fueron:
    - Centura (Clientes Windows)
    - Oracle (Servidor de Datos)
    - Acu Cobol (Clientes del Sistema de Caja)
    - ASP (Cliente Web sobre IIS)
    - SAP.(Producto empaquetado)
- Definir las principales aplicaciones que servirían de prototipo para realizar la integración.
- Evaluación de los resultados obtenidos.

Finalmente, luego de evaluar el plan piloto se optó por BEA Tuxedo como una infraestructura de integración Tecnológica.

4.4 ARQUITECTURA TECNOLÓGICA DE BELLSOUTH SIN TUXEDO

A continuación, la figura 4.4-1 muestra la arquitectura de las principales aplicaciones que posee Bellsouth. Se puede apreciar a simple vista que hay tan solo un punto de integración entre ellas y que lo constituye el Sistema de Bases de Datos Relacional Oracle. Se puede observar además que la aplicación de SAP es un elemento aislado por completo.

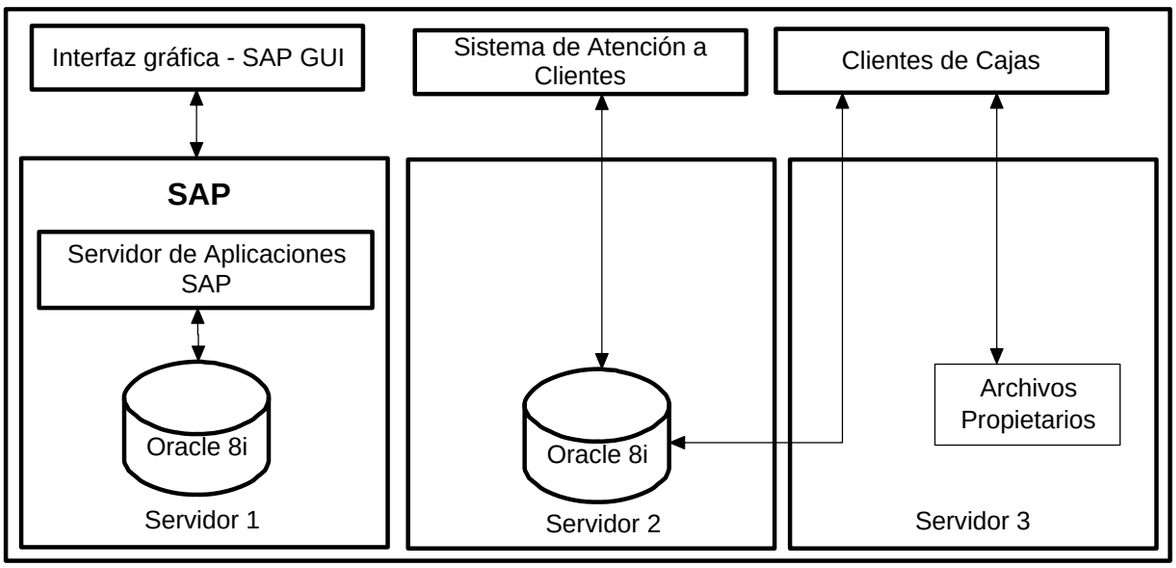


Figura 4.4-1. Arquitectura de las Aplicaciones sin Tuxedo.

Los siguientes puntos ayudan a entender la figura:

- SAP constituye uno de los elementos más importantes de la empresa porque implementa la mayor parte de la funcionalidad de los departamentos. Está compuesto además de una interfaz gráfica (SAP GUI) que permite hacer uso del sistema y de un servidor de aplicaciones con su propio motor de datos Oracle.
- El Sistema de Atención a Clientes es una aplicación escrita en Centura 1.5. Es la cara visible de las asistentes de venta. Se ejecuta sobre plataforma Windows y está implementada bajo un esquema Cliente/servidor de dos capas. Su servidor de datos Oracle se ejecuta en un servidor con HP-UX 11.
- Los clientes de Caja, son aplicaciones ACUCOBOL que se ejecutan sobre máquinas Linux. Estas aplicaciones poseen su propio esquema de seguridad y se ejecutan en forma independiente de las dos anteriores.

Esta aplicación es la única que se integra con el sistema de Cliente a nivel de datos.

4.5 DISEÑO

4.5.1 Consideraciones Básicas

Herramientas de Desarrollo Cliente

- Centura Team Developer (CTD) 1.5, es un Ambiente de Desarrollo Integrado (IDE) para desarrollar aplicaciones de negocio estratégicas bajo el sistema operativo Windows. Este producto se utiliza en la compañía porque ofrece un ambiente de desarrollo orientado a objetos y además por proveer conectividad nativa a diversas bases de datos (Oracle, SQLBase, DB2, SQL Server, Sybase, etc.).
- Estaciones de Trabajo Pentium III con Sistema Operativo Windows 2000 Profesional.
- Para desarrollo de clientes Web se utiliza la tecnología Active Server Pages de Microsoft que se ejecutan bajo el servidor Web IIS 5 sobre una máquina con sistema operativo Microsoft Windows 2000 Advanced Server.

Herramientas de Desarrollo Servidor

- Lenguaje de programación HP C que es la versión de Hewlett-Packard del lenguaje de programación C que se utiliza sobre máquinas HP 9000 con sistema operativo HP-UX 11. Para compilar los Servidores de Aplicación Tuxedo escritos en C se utiliza el utilitario make.
- Servidor de Datos Oracle 8i sobre Sistema Operativo HP-UX 11.
- DAG 7.0, es un generador de código de ORDEN utilizado en este proyecto para generar los procedimientos almacenados de Oracle. La ventaja de esta herramienta es que maneja centralizadamente el código y los campos, por cuanto, puede ser regenerado en cualquier instante.

Aplicaciones Seleccionadas

Las aplicaciones que se integraron inicialmente en Bellsouth fueron determinadas en base a los requerimientos más urgentes de la compañía. Donde esencialmente había que lograr intercambiar información con SAP, el

cual era hasta antes de la integración una caja negra. De ahí que se buscó un adaptador que pudiese hablar con SAP.

Respecto de las aplicaciones involucradas se optó por las siguientes:

- Consulta de la Cuenta Corriente de un Cliente, debía ser implementado como una opción más del Sistema de Atención a Clientes construido en Centura.
- Implementar la misma consulta de cuenta corriente a través de una interfaz WEB.
- Proveer la lógica asociada a los procesos de negocio de Reposición y Recarga como servicios Tuxedo cada vez que se registraban pago en el sistema de Cajas construido en Acu-Cobol.

Tipo de Mensaje Tuxedo

Se definió trabajar con el tipo de buffer Tuxedo FML32, por cuanto ofrece el mayor grado de flexibilidad en el intercambio de mensajes. Adicionalmente, el adaptador **BEA eLink Adapter** (definido en el punto 4.5.3) usa sólo buffers FML32 para comunicarse con SAP.

4.5.2 Protocolo de Mensajería

Se definió que cada mensaje que se intercambie entre aplicaciones clientes y servidores debe respetar un protocolo de control estándar para el manejo de errores y otros datos de interés.

Adicionalmente cada campo de un servicio Tuxedo debe tener las siguientes propiedades.

- Un **Nombre** que represente el significado del campo.
- Un **Modo** que indica la dirección donde va el campo, los valores posibles son:
    - Modo *Input*, para aquellos campos que son de entrada al servicio y almacenan un único valor.
    - Modo *Output*, para aquellos campos que son de salida del servicio y almacenan un único valor.
    - Modo *InputOutput*, que son de entrada y salida al servicio y almacenan un único valor.
    - Modo *ColumnInput*, para definir un conjunto de datos de entrada a un servicio. Por ejemplo, un cliente puede enviar una lista de 3

registros con los campos Rut y Dirección para actualizar la dirección de cada rut que va de entrada [{1-9 ,Dirección1}; {2-7 , Dirección2};{123-5,Dirección3}].

- Modo *ColumnOutput*, para definir un conjunto de registros salida del servicio Tuxedo, por ejemplo, obtener la lista completa de los campos Código y Glosa de las regiones de Chile.

- Un **FieldId** es un identificador de field utilizado por Tuxedo para reconocer un campo específico dentro de un mensaje FML32. Los FieldId utilizados en la compañía se definieron en una única tabla llamada TblGlobal.fld, donde reside todo el universo de campos que se intercambian en los mensajes. Ver Anexo No 4-3.

La siguiente tabla muestra el modo y tipo de dato que posee cada campo del protocolo propuesto.

Nombre del Campo	Modo	Tipo	Largo	Valor por Defecto
fld_hdr_NroFilasCI	input	short	[0,9999]	0
fld_hdr_MaxFilasCO	input	short	[0,9999]	500
fld_hdr_NroFilasCO	output	short	[0,9999]	0
fld_hdr_CodRetorno	output	short	[0-1]	1
fld_hdr_MoreData	output	short	[0-1]	0
fld_hdr_MsgCode	output	long	[0-38000-58000-68000]	0
fld_hdr_MsgText	output	char *	[0,250]	“ “

La siguiente tabla describe el significado de cada campo.

Nombre del Campo	Significado
fld_hdr_NroFilasCI	Es un número que indica la cantidad de filas o registros que está enviando el cliente al servicio. Si no fuese el caso, se debe mandar 0.
fld_hdr_MaxFilasCO	Este campo indica el número máximo de filas que puede recibir un cliente cuando hay conjuntos de filas resultantes. Este valor lo especifica el cliente, y se debe basar en la cantidad máxima de filas que es capaz de procesar o la cantidad de filas que el usuario final es capaz de analizar.

	Posibles valores son [100, 200, 300, 400, **500,** 1000,3000,5000]. Siempre se debe enviar aun cuando no hay filas resultantes.
fld_hdr_NroFilasCO	Este valor siempre lo actualiza el servidor y contiene el número de filas resultantes. Normalmente este número es menor o igual a fld_hdr_MaxFilasCO. El cliente deberá iterar de 0 a fld_hdr_NroFilasCO -1 para obtener cada uno de los registros de respuesta. Observe que este campo podrá tener un valor distinto de 0 sólo cuando el servicio posea parámetros de COLUMNOUTPUT, en caso contrario no será necesario validar su contenido.
fld_hdr_CodRetorno	Este valor lo devuelve el servidor con 1 en caso de éxito y 0 en caso de error. Actualmente siempre se devuelve 1.
fld_hdr_MoreData	Este campo indica si hay más filas por retornar. Este campo tiene sentido cuando fld_hdr_MaxFilasCO = fld_hdr_NroFilasCO < Universo Total de Registros de respuesta. 1, hay más filas 0, no hay más filas.
fld_hdr_MsgCode	Código de error que siempre envía el servidor, existen 4 niveles de error en el servidor: 0: La ejecución del servicio Tuxedo fue exitosa. 38000: Mensaje de Información. 58000: Mensaje de Advertencia. 78000: Mensaje de Error. Este es el primer campo que se debe validar para saber el resultado de la llamada a un Servicio Tuxedo.
fld_hdr_MsgText	Este es el texto representativo del error que siempre envía el servidor y podrá ser tan específico como se desee. Cuando no hay error su valor es “ ”.

A continuación, la figura 4.5-1 muestra gráficamente el intercambio de mensajes FML32 entre un cliente Tuxedo y un servidor Tuxedo.

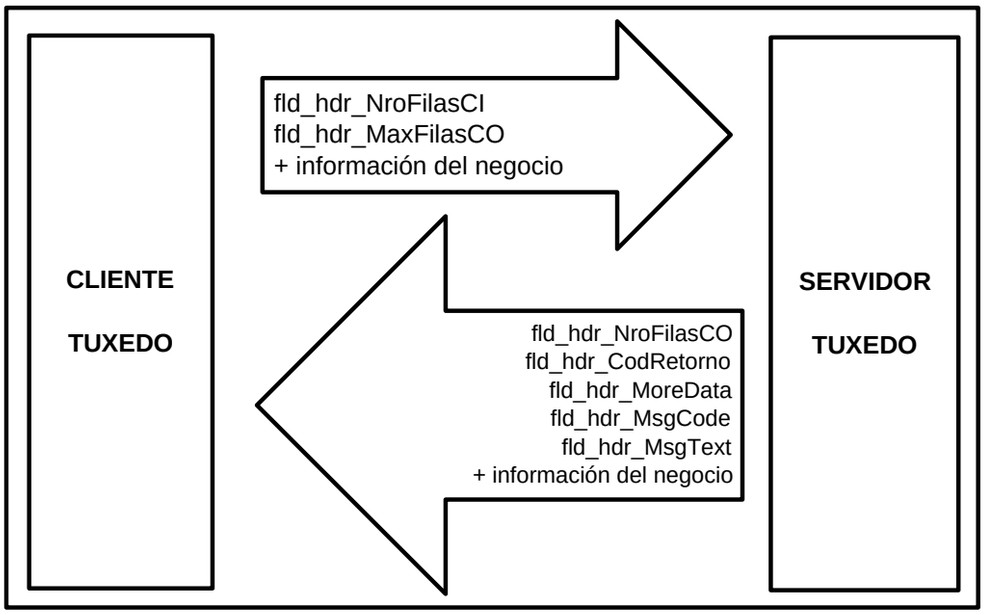


Figura 4.5-1. Intercambio de mensajes FML32.

4.5.3 Adaptador de Integración con SAP

Un producto de BEA llamado **BEA eLink Adapter for R/3 BAPI/RFC** fue seleccionado como la infraestructura de integración con el ambiente SAP. BEA eLink trabaja en conjunto con BEA Tuxedo y permite la implementación de aplicaciones de misión crítica y de acceso en tiempo real a las transacciones, datos y funciones de SAP. BEA eLink está constituido por Servidores de Aplicación genéricos que pueden ser personalizados de acuerdo a las necesidades del cliente. Funcionalmente, lo que hacen los servidores de aplicación genéricos de BEA eLink es exportar como servicios Tuxedo las RFCs y BAPIS de SAP. Para ello, es fundamental poseer la definición formal de las RFCs y BAPIs, es decir, nombres de la función de negocio, parámetros de entrada y salida con sus respectivos tipos. De esta manera, cualquier cliente Tuxedo tiene la capacidad de llamar a las RFCs y BAPIs que están representadas a través de servicios Tuxedo exportados en un dominio adicional.

4.5.4 Arquitectura de Integración con Tuxedo

Por último, se mostrará la arquitectura integrada a través de Dominios Tuxedo. Se definieron dos dominios:

- El dominio **bellsouth**, que físicamente reside en el servidor dos (ver Figura 4.5-3) y será considerado el dominio principal donde se implemente la lógica de negocio de Bellsouth. Aquí estarán los servicios que deberán llamar los clientes. Además, este servidor hará requerimientos al dominio elink cada vez que necesite una función de negocios de SAP.
- El dominio **elink**, residirá físicamente en el servidor 1, es decir, junto al servidor de aplicaciones de SAP. Este dominio sólo exportará los servicios Tuxedo asociados a RFCs y BAPIs a través de los servidores genéricos.

Desde un punto de vista lógico, las aplicaciones clientes que necesiten acceder a la lógica de negocios de SAP deberán hacerlo llamando a servicios Tuxedo que residen en el dominio Bellsouth. Por otro lado, los clientes SAP GUI seguirán accediendo a SAP en forma directa. Lo que se ha hecho es exportar la lógica de negocio a través de una capa intermedia constituida por servicios Tuxedo, lo que trae como beneficio que dichos servicios se pueden reutilizar desde cualquier aplicación cliente.

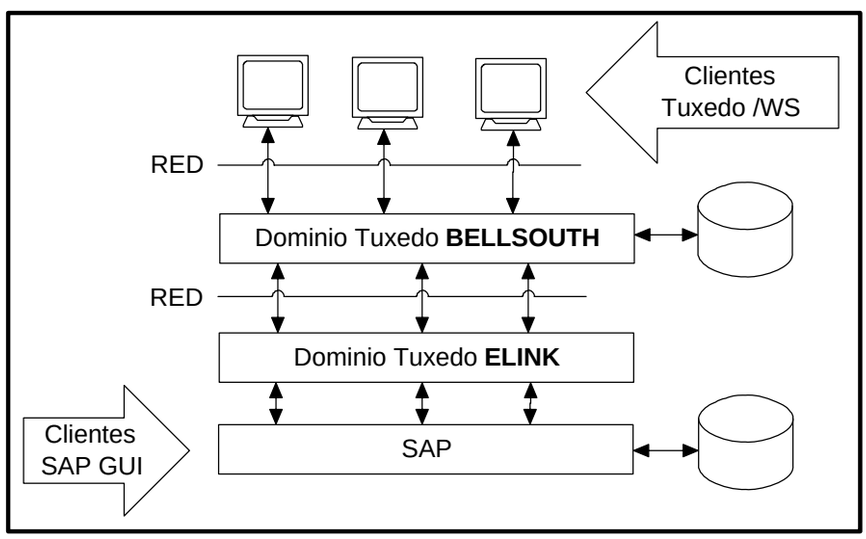


Figura 4.5-2. Arquitectura General de Integración

En términos más específicos, se puede observar en la figura 4.5-3 que el dominio Elink conoce el protocolo para llamar a las RFCs y BAPIs de SAP y, a partir de eso, es capaz de exportar dicha funcionalidad como servicios Tuxedo. Adicionalmente, se determinó que este dominio se dedique sólo a exportar las funciones de SAP y no otra lógica. Por otro lado, se estableció que sea el dominio Bellsouth quien actúe como un único punto de entrada para todos los clientes y cualquier lógica de negocios nueva, que involucre el acceso a la base de datos Oracle de Bellsouth(Servidor 2), acceso a los servicios de SAP o a otro tipo de recursos, se implemente en este dominio.

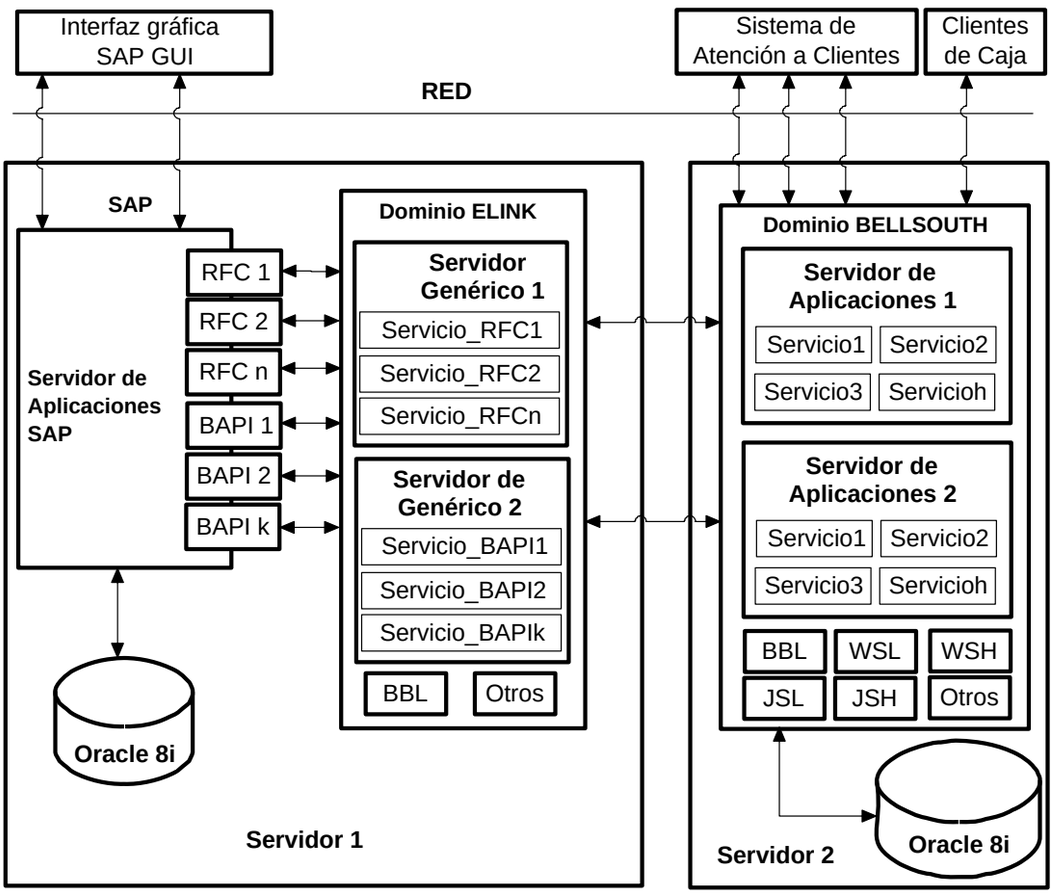


Figura 4.5-3. Arquitectura Específica de Integración

4.6 IMPLEMENTACIÓN

4.6.1 Clientes Tuxedo

Respecto a la implementación de los clientes, Bellsouth tiene por política encargar el desarrollo de las aplicaciones clientes a personal externo, por lo cual, el trabajo que aquí se presenta sólo define los mecanismos de

comunicación específicos para un ambiente de desarrollo o lenguaje de programación determinado. El objetivo es mantener al desarrollador alejado de la problemática de comunicaciones y, de este modo, sólo se dedican al desarrollo de la aplicación en sí.

Aplicación de Consulta de Cuenta Corriente en Centura

En este caso se optó por diseñar e implementar dos clases Centura; una clase que encapsule la lógica asociada al manejo de la conexión y desconexión de un dominio Tuxedo y otra que permita el llamado a un servicio Tuxedo, lo cual evita que el desarrollador se vea involucrado en el manejo de memoria para los buffers FML32.

Las dos clases que se implementan son:

- cTux_Session, maneja la conexión y desconexión de un dominio Tuxedo.
- cTux_Service, maneja el llamado a un servicio Tuxedo.

Para ver más detalles, ver Anexo No 4-2.

Ejemplo de uso:

Dadas dos variables definidas como sigue:

- cTux_Session: *tuxSession*
- cTux_Service: *tuxService*

La secuencia lógica para conectarse y llamar a un servicio es la siguiente:

tuxSession.**connect**

tuxService.**initService**

tuxService.*addField<type>*

tuxService.*addField<type>*

tuxService.*addField<type>*

tuxService.**callService**

tuxService.*getField<type>*

tuxService.*getField<type>*

tuxService.*getField<type>*

tuxService.**termService**

tuxSession.**disconnect**

Donde <type> corresponde al tipo de datos que se desea manejar de acuerdo a los buffers FML32.

La figura 4.6-1 muestra el detalle de los componentes involucrados en el desarrollo del cliente Centura:

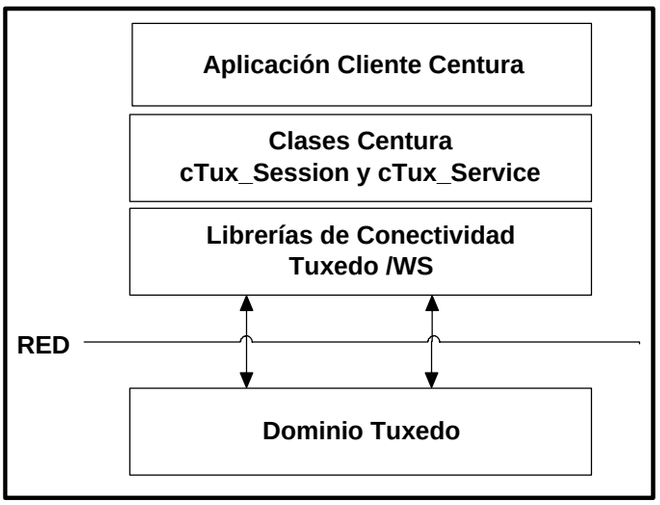


Figura 4.6-1. Componentes del Cliente Centura.

Lo importante en el diseño de esta solución es que el programador de aplicaciones cliente ve una interfaz más simple a través de las clases y, son éstas, las que trabajan en forma directa con las DLLs de conectividad Tuxedo. Además las clases manejan la complejidad asociada al manejo de buffers FML32.

CLASE cTux_Session

A continuación se muestra la lista de métodos y propiedades implementado por la clase:

- Método **connect**
- Método **disconnect**
- Propiedad **getErrorCode**
- Propiedad **getErrorText**
- Propiedad **getUser**
- Método **ulog**

CLASE cTux_Service

Esta clase permite llamar a un servicio Tuxedo y recuperar los datos. A continuación se muestra la lista de métodos y propiedades disponibles.

- Método **initService**
- Método **callService**
- Método **termService**
- Método **existsField**
- Método **tranBegin**
- Método **tranCommit**
- Método **tranRollback**
- Propiedad **getMessageCode**
- Propiedad **getMessageText**
- Propiedad **getNroFilasCO**
- Propiedad **setMaxFilasCO**
- Propiedad **getMoreData**
- Propiedad **setNroFilasCl**
- Método **addFieldLong**
- Método **addFieldShort**
- Método **addFieldString**
- Método **addFieldFloat**
- Método **addFieldDouble**
- Método **addFieldChar**
- Propiedad **getFieldLong**
- Propiedad **getFieldShort**
- Propiedad **getFieldString**
- Propiedad **getFieldFloat**
- Propiedad **getFieldDouble**
- Propiedad **getFieldChar**
- Propiedad **getErrorCode**
- Propiedad **getErrorText**
- Método **ulog**

Respecto a los servicios Tuxedo que esta aplicación utiliza, ahora sólo se nombrarán y en la sección de servidores se detallarán con más detalle

- c_PartidaS
- c_DetPartidaS

Aplicación de Cajas en AcuCobol

En este caso se implementó una solución con una filosofía similar a la del cliente Centura, es decir, se escribió una librería en lenguaje C que tuviese toda la lógica asociada a la conexión al dominio Tuxedo y la llamada a los servicios Tuxedo que se necesitaban. Por tanto, se escribió una función por cada servicio Tuxedo que se necesitaba llamar. De este modo, el desarrollador de la aplicación en AcuCobol sólo tenía que llamar a dos funciones sin mayores inconvenientes técnicos.

Para integrar aplicaciones AcuCobol con otras aplicaciones que ejecutan código nativo (típicamente aplicaciones escritas en C), es necesario incorporar en el *runtime* de AcuCobol las funciones externas que la aplicación Cobol necesita. En particular, esta sección describe como integrar aplicaciones AcuCobol con BEA Tuxedo /WS.

Los servicios que debe llamar la aplicación de caja son:

- j_DocCtaCteS
- j_AnalizaDeudaI

Especificación Servicio j_DocCtaCteS

Obtener los documentos de un cliente-cuenta, dados los parámetros Rut o Número Servicio o Número Documento.

Parámetros	Descripción	Modo
fld_TipoConsultaCj	Tipo Consulta de Caja	Input
fld_CodEmpresaFact	Código Empresa Facturadora	Input
fld_RutCliente	Rut Cliente	Input
fld_NroServicio	Número Servicio	Input
fld_TipoDocCaja	Tipo Documento	Input
fld_FolioDocumento	Folio Documento	Input
fld_NroCliente	Número Cliente	Input
fld_NroCuenta	Número Cuenta	Input
fld_SapId	Sap Id	Input

fld_TipoDocCaja	Código Tipo documento	ColumnOutput
fld_GlosaTipoDoc	Glosa tipo documento	ColumnOutput
fld_CodEmpresaFact	Código empresa facturadora	ColumnOutput
fld_FolioDocumento	Folio Documento	ColumnOutput
fld_CodBanco	Código del banco de cheque protestado	ColumnOutput
fld_NroCliente	Número Cliente	ColumnOutput
fld_NroCuenta	Número Cuenta	ColumnOutput
fld_NroServicio	Número Servicio	ColumnOutput
fld_Saldo	Saldo	ColumnOutput
fld_EstadoCteCta	Estado del cliente cuenta en ISR	ColumnOutput
fld_IndDocCartera	Indica si es documento en cartera	ColumnOutput
fld_FechaVctoPago	Fecha vencimiento o fecha pago	ColumnOutput
fld_IndCtaControlada	Indica si es cuenta controlada	ColumnOutput
fld_FolioInternoSap	Folio Documento en SAP	OutputColumn
fld_SapId	Identificación Cliente-Cuenta en SAP	ColumnOutput
fld_ValorCCO	Valor Cuenta Controlada	ColumnOutput
fld_ValorImporte	Importe	ColumnOutput
fld_NombreCliente	Nombre Cliente	ColumnOutput
fld_NumeroRutCliente	Número del Rut del Cliente	ColumnOutput
fld_DVRutCliente	Dígito Verificador del Rut Cliente	ColumnOutput

Especificación Servicio j_AnalizaDeudaI

Decide a partir de los datos de entrada más la información del estado en la base de datos Oracle y la deuda en SAP, si al cliente-cuenta deberá hacérsele reposición y/o recarga.

Parámetros	**Descripción**	**Clase**
fld_NroCliente	Número Cliente	ColumnInput
fld_NroCuenta	Número Cuenta	ColumnInput
fld_SapId	Sap Id	ColumnInput
fld_IndCtaControlada	Indicador Cuenta Controlada	ColumnInput
fld_NroServicio	Número Servicio	ColumnInput
fld_CodEmpresaFact	Código Empresa Facturadora	ColumnInput
fld_Sucursal	Sucursal	ColumnInput
fld_Caja	Caja	ColumnInput
fld_Host	Host	ColumnInput
fld_FechaRecaudacion	Fecha de Recaudación	ColumnInput
fld_TransDia	Número Transacción Diaria	ColumnInput
fld_TransHist	Número Transacción Histórica	ColumnInput
fld_TipoDocCaja	Tipo Documento	ColumnInput

fld_FolioDocumento	Folio Documento	ColumnInput
fld_MontoRecaudado	Monto Recaudado	ColumnInput
fld_Observacion	Observación	ColumnInput

La arquitectura de la solución es la que se muestra en la figura 4.6-2:

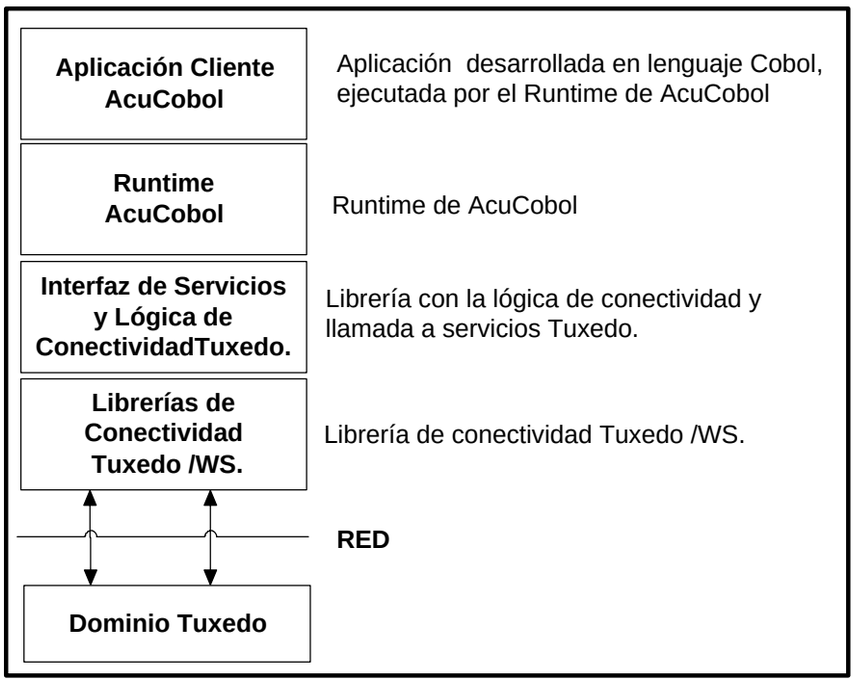


Figura 4.6-2. Arquitectura de Integración con AcuCobol.

La forma de modificar el runtime del AcuCobol es el siguiente:

El archivo **direct.c** (parte del runtime) debe registrar todas las funciones externas al AcuCobol que se invocarán desde el código Cobol de las aplicaciones. Para ello todas las funciones externas se deben registrar en la siguiente estructura:

```
struct  DIRECTTABLE WNEAR LIBDIRECT[] = {
    { NULL,             NULL,       0 }
    };
```

En nuestro caso, por ejemplo, si desde aplicaciones Cobol se llama a las funciones **j_DocCtaCteS** y **j_AnalizaDeudaI** escritas en C y que implementan la interfaz para llamar a servicios provistos por servidores de aplicación Tuxedo, se debe hacer de la siguiente forma:

```
extern int j_DocCtaCteS();
```

```
extern int j_AnalizaDeudaI();
struct  DIRECTTABLE WNEAR LIBDIRECT[] = {
        { "J_DOCCTACTES",      FUNC j_DocCtaCteS,    C_int },
        { "J_ANALIZADEUDAI",   FUNC j_AnalizaDeudaI, C_int },
        { NULL,                NULL,         0 }
        };
```

En donde el primer parámetro de cada entrada describe el nombre con el cual será conocida la función en Cobol, este nombre debe estar en mayúsculas. El segundo parámetro es el nombre de la función C y, el tercer parámetro, es el tipo de dato retornado por la función, en este caso es un *integer*. En este caso la función C: **j_DocCtaCteS** es conocida en el ambiente Cobol con el nombre **J_DOCCTACTES**.

4.6.2 Servidores Tuxedo

Como se mencionó anteriormente todos los servidores deben ajustarse al protocolo establecido entre clientes y servidores. Lo anterior permitirá un control adecuado de los errores de Oracle y Tuxedo.

A continuación la figura 4.6-3 mostrará la secuencia lógica de los servidores construidos en la compañía.

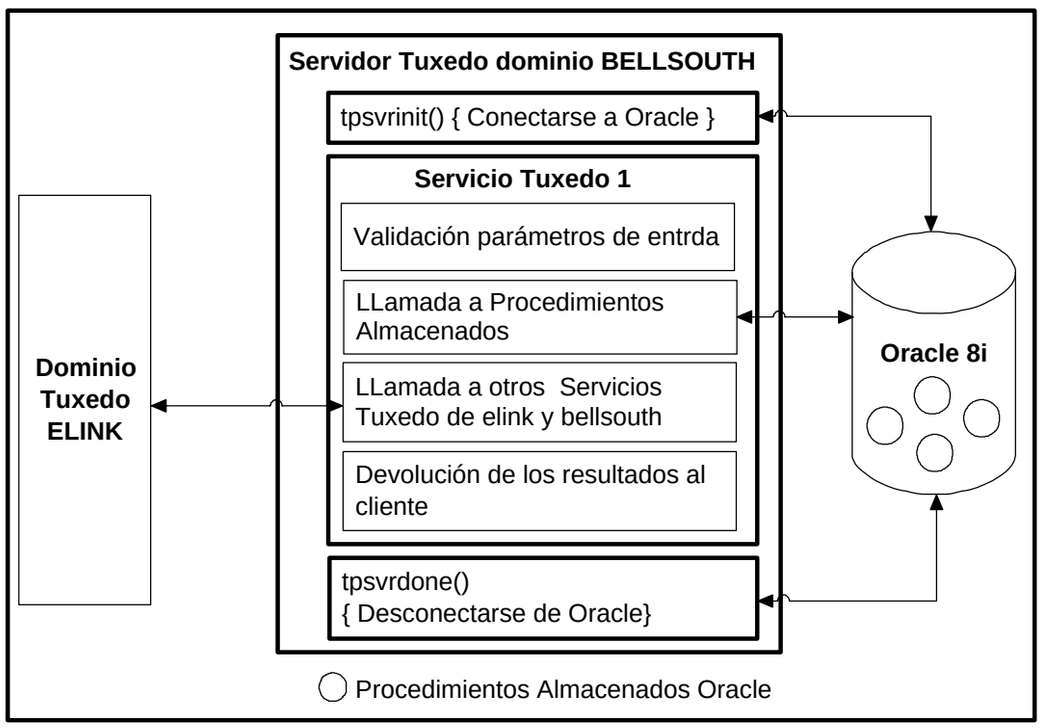


Figura 4.6-3. Secuencia lógica de los servidores.

De la figura se pueden deducir varias cosas:

- Se utiliza una conexión a la base de datos que dura mientras el servidor de aplicaciones Tuxedo esté en ejecución. El servidor de aplicaciones Tuxedo se conecta a Oracle al inicio y se desconecta al terminar su ejecución.
- Se establece que los servidores de aplicación Tuxedo accedan a la base de datos Oracle sólo a través de procedimientos almacenados.
- Resulta natural pensar que los servidores de aplicación que acceden a la base de datos Oracle, necesariamente se deben compilar con las librerías de conectividad de dicha base de datos. Por lo tanto, el ejecutable final está compuesto por el código con la lógica de negocio, las librerías del motor de datos y las librerías propias de Tuxedo.
- Otro aspecto de los servidores Tuxedo es que pueden invocar a servicios que pertenezcan a otros servidores del mismo dominio o de dominios distintos, como es el caso en que se accede a la funcionalidad de SAP a través del dominio elink.

Dominio Tuxedo Bellsouth

El dominio bellsouth constituye en términos de configuración un dominio normal, sus principales elementos son:

- Archivo de configuración principal **bellsouth.ubb**, ver Anexo No 4-3.
- Archivo de configuración para integrarse con otros dominios **bellsouth.dom,** ver Anexo No 4-4. En este archivo se especifican los otros dominios y los servicios que este dominio exporta o importa.
- Servidores de Aplicación construidos en la compañía.
    - sCtaCte, servidor de consultas de cuenta corriente, ver Anexo No 4-6.
    - sCaja, servidor de cajas, ver Anexo No 4-7.
- Servidores Administrativos propios de Tuxedo.
    - BBL, maneja la información de Tuxedo en memoria.
    - WSL y WSH, manejan las conexiones clientes Tuxedo /WS.
    - JRAD, JSL y JSH, manejan las conexiones de clientes WEB.
    - JREPSVR, maneja el repositorio de Jolt 1.2.

- DMADM, GWADM, GWTDOMAIN, manejan la comunicación con otros dominios, en este caso con el dominio elink.

Dominio Tuxedo Elink

El objetivo principal de este dominio es exportar la funcionalidad de SAP a través de servicios Tuxedo. Para lo cual está constituido de los siguientes elementos:

- Archivo de configuración principal elink.ubb, ver Anexo No 4-5.
- Archivo de configuración elink.dom para integrarse con otros dominios. En este archivo se especifican los servicios Tuxedo que se exportan al dominio bellsouth.
- Servidores genéricos que exportan la funcionalidad SAP. Hay dos servidores:
  - cr3rfcin, este servidor es el encargado de llamar a RFCs y BAPIs de SAP y exportar dicha funcionalidad como un servicio Tuxedo. En otras palabras permite hacer requerimientos hacia SAP.
  - cr3rfcout, este servidor permite hacer llamados desde SAP a servicios Tuxedos. Su lógica es recibir requerimientos desde SAP y llamar a los servicios Tuxedo.

Hay que indicar que estos dos programas trabajan en base a archivos de configuración donde se especifica la relación entre un nombre de RFC o BAPI y un nombre de servicio Tuxedo.

A continuación se verá el ejemplo de configuración de una RFC que necesita ser exportada como un servicio Tuxedo. Esta RFC se utiliza en los servicios de Caja.

Paso 1: Especificación de la RFC que obtiene el estado del Cliente–Cuenta de SAP:

Nombre	Z_TX_CTACTECAJA
Parámetros de entrada	FLD_TIPOCONSULTACJ FLD_SOCIEDAD FLD_RUT FLD_TIPODOCCAJA FLD_FOLIODOCUMENTO FLD_CLIENTE FLD_CUENTA FLD_IDSAP
Tablas de Entrada	No hay tablas de entrada en la RFC.
Parámetros de salida	RETURN
Tablas de Salida	SALIDA

Paso 2: Definir el archivo de configuración que asocie la RFC al servicio Tuxedo. Este archivo será utilizado posteriormente por el servidor genérico cr3rfcin.

Se establece como norma que el nombre del archivo tenga la siguiente forma:

cr3rfcin_<NombreServidortuxedo>.env

donde:

<NombreServidortuxedo>, corresponde al módulo principal que utiliza este servicio o que generó su construcción.

La extensión del archivo .env viene de environment, porque el servidor genérico trata estas configuraciones como variables de ambiente.

En nuestro caso, el archivo se llamará **cr3rfcin_ctactecajas.env** y parte de su contenido es:

SERVICE_LIST=Tx_jDocCtaCte

[SERVICE=Tx_jDocCtaCte**]**
CR3_RFC_NAME=Z_TX_CTACTECAJA
CR3_IMPORT_PARAMS=FLD_TIPOCONSULTACJ,FLD_SOCIEDAD,FLD_RUT,FLD_TIP
ODOCCAJA,FLD_FOLIODOCUMENTO,FLD_CLIENTE,FLD_CUENTA,FLD_IDSAP
CR3_IMPORT_TABLES=
CR3_EXPORT_PARAMS=RETURN
CR3_EXPORT_TABLES=ZSALIDA
CR3_RFC_ROW_DETAIL=Y

Para ver el contenido completo del archivo ver Anexo No 4-8.
El elemento **SERVICE_LIST** corresponde a una lista separada por comas de los nombres de servicios Tuxedo que se exportarán en este archivo de configuración. Por cada servicio Tuxedo que exista en esta lista, deberá existir una entrada de la forma [SERVICE=<NombreServicioTuxedo>], en nuestro ejemplo tenemos [SERVICE=Tx_jDocCtaCte] y en esta sección se especifican los parámetros formales que posee la RFC(parámetros de entrada y salida). Por último, hay que hacer notar que los campos que aquí se indican corresponden a FieldsId que existen en la tabla global de fields(FML32) que utiliza el dominio elink.

5 CONCLUSIONES

Las conclusiones que aquí se presentan se basan en los resultados obtenidos en el desarrollo de esta tesis y en la experiencia lograda a partir de la integración realizada en Bellsouth.

- El problema de integración debe ser abordado como un problema global de la empresa y no como un simple problema técnico asociado a un conjunto de aplicaciones.
- Desde un punto de vista general, la integración de aplicaciones aisladas en una empresa permite presentar a los usuarios una visión unificada de la información, permitiendo así tomar mejor decisiones
- Se realizó la integración de diversos sistemas en Bellsouth a través de la metodología propuesta en esta tesis.
- Se introdujo en Bellsouth el concepto de Tuxedo como un marco de trabajo que permite la integración ordenada de todos los sistemas de la compañía.
- Se definieron estándares de análisis, diseño e implementación que permiten a la compañía ejecutar sus propios proyectos sin la necesidad de personal externo especializado.
- Se logró a través del uso de los servicios Tuxedo la reutilización de la lógica de negocios de manera transaccional entre diversas aplicaciones.
- Se demostró que con la integración de sistemas se pueden crear procesos de negocio más eficientes que permiten dar a los clientes finales un mejor servicio.
- La introducción de la tecnología Tuxedo por primera vez en una compañía requiere de personal especializado tanto para tareas de administración como para tareas de desarrollo de proyectos. Adicionalmente, con el paso del tiempo, se deberán crear al interior de la compañía nuevos roles que asuman estas responsabilidades.
- La implementación de la tecnológica Tuxedo permitirá soportar crecimientos y cambios rápidos de manera que la empresa pueda reaccionar en forma oportuna. Lo anterior da una ventaja de adecuación a las nuevas tecnologías y permanencia en el mercado.

- Se logró establecer los mecanismos que permiten hacer pruebas de esfuerzo y análisis de los tiempos de respuesta en los servidores Tuxedo.

Bibliografía

[1] Your Blueprint for Building a Successful EAI Solution.
http://www.ejiva.com/n_whitepaperb.htm

[2] Consensual Trends for Optimizing the Constitution of Middleware.
Stephane Spahni, Jean Raoul Scherrer, Dominique Sauquet, Pier Angelo Sottile.

[3] Middleware – The Essential Component for Enterprise Client/Server Applications.
International Systems Group, Inc.

[4] Concept : Middleware and EDI Data.

[5] La importancia de un Middleware Robusto y Escalable en las Soluciones Empresariales Cliente/Servidor.
Victor Manuel Díaz

[6] Message Oriented Middleware (MOM)
Markku Korhonen

[7] Implementing Remote Procedure Calls
Andrew D. Birrell, Bruce Jay Nelson
Xerox Palo Alto Research Center

[8] Comparing Remote Procedure Calls
John Barkley, Octubre de 1993
http://hissa.nist.gov/rbac/nistir/5277/titlerpc.html

[9] Aplicaciones Distribuidas sobre Arquitectura Cliente Servidor utilizando Tuxedo.
Tesis Universidad Austral de Chile, Ingeniería Civil en Informática, 1998.
Enrique D. Medina O.

[10] Building Client/Server Applications Using Tuxedo.
Carl L. Hall.

[11] Transaction Processing Monitor,
Communications of the ACM, Noviembre 1990.
Philip A. Bernstein.

[12] The Tuxedo System.
Juan M. Andrade.
Mark T. Carges.
Terence J. Dwyer.
Stephen D. Felts.

ANEXO No 4-1

TABLA DE FIELDS DEL DOMINIO BELLSOUTH

```
#
#   Tabla de fields para implantacion de Tuxedo en Bellsouth.
#
# ============================================================
# Conjunto de campos que componen el header de un mensaje tuxedo.
# ============================================================
*base  100
# name                  number   type    flags   comments
fld_hdr_Username        1        string  -       Username del cliente
fld_hdr_NroFilasCI      2        short   -       Numero de Filas de entrada en un Set
fld_hdr_NroFilasCO      3        short   -       Numero de Filas de salida  en un Set
fld_hdr_MaxFilasCO      4        short   -       Maximo Numero de Filas de salida
fld_hdr_CodRetorno      5        short   -       Cod. Retorno del Servicio Tux. Exito=1/Error=0
fld_hdr_MoreData        6        short   -       Hay mas datos=1, No hay mas datos = 0
fld_hdr_MsgCode         7        long    -       Codigo del Mensaje de Oracle ORA-XXXX
fld_hdr_MsgText         8        string  -       Texto  del Mensaje de Oracle
#
#
# ============================================================
# campos asociados al los servicios de consulta cta cte.
# ============================================================
*base  1024
# name                  number   type      flags   comments
# ------------------------------------------------------------------------
fld_NroCliente          1        string    -       Cliente
fld_NroCuenta           2        string    -       Cuenta
fld_SapId               3        string    -       Sap ID
fld_CodEmpresaFact      4        string    -       Codigo empresa facturadora.
fld_TipoConsulta        5        char      -       Tipo Consulta [A: abiertas, T: todas]
#
fld_FolioInternoSap     6        string    -       DOC_NO - Sap - char(10)
fld_IndOperacionCtaEsp  7        char      -       IndOperacionCtaEsp
fld_FechaEntradaDocCont 8        string    -       FechaEntradaDocCont
fld_FechaEmision        9        string    -       FechaEmision
fld_DocRef              10       string    -       DocRef
fld_TipoDoc             11       string    -       TipoDoc
fld_GlosaTipoDoc        12       string    -       GlosaTipoDoc
fld_IndicadorDH         13       char      -       IndicadorDH
fld_GlosaDetalle        14       string    -       GlosaDetalle
fld_FechaVctoPago       15       string    -       FechaVctoPago
fld_Debe                16       double    -       Columna DEBE
fld_Haber               17       double    -       Columna HABER
fld_FolioDocumento      18       string    -       Alloc_NMBR Sap char(18)
fld_Saldo               19       double    -       Saldo de Stma. CCC formato DOUBLE
#
# Estos campos son del RFC de Consulta Cta Cte, que no son manejados
# directamente con la interfaz de los servicios o clientes tuxedo.
#
fld_SapSaldo            20       string    -       Saldo de Stma. SAP formato STRING
fld_SapFechaDesde       21       string    -       Fecha Desde para RFC ConCtaCte
fld_SapFechaHasta       22       string    -       Fecha Hasta para RFC ConCtaCte
fld_SapDebeHaber        23       string    -       Campo AMOUNT que va al Debe o Haber
#
#
fld_AnoEjercicio        24       string    -       Ano ejercicio en formato String
fld_Posicion            25       short     -       Posicion dentro del detalle
fld_Importe             26       double    -       Importe del detalle
```

ANEXO No 4-2
DEFINICIÓN DE CLASES CENTURA

CLASE cTux_Session

Método **connect**

- *Objetivo*

Este método permite conectarse a Tuxedo.

- *Retorno*

Boolean: TRUE cuando la conexión a Tuxedo fue exitosa, FALSE cuando hay error.

- *Parámetros de Entrada*

String: newUsername, usuario que se conecta.
String: newPassword, password del usuario.
String: newClient, nombre aplicación cliente.

- *Parámetros de Salida*

No hay.

Método **disconnect**

- *Objetivo*

Este método permite desconectarse de Tuxedo.

- *Retorno*

Boolean: TRUE cuando hay éxito, FALSE cuando hay error.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getErrorCode**

- *Objetivo*

Esta propiedad permite obtener el código de error Tuxedo.

- *Retorno*

Number: número del error Tuxedo.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getErrorText**

- *Objetivo*

Esta propiedad permite obtener el texto del error Tuxedo.

- *Retorno*

String: texto descriptivo del error Tuxedo.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getUser**

- *Objetivo*

Esta propiedad permite obtener el username conectado.

- *Retorno*

String: username conectado a Tuxedo.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Método **ulog**

- *Objetivo*

Este método permite escribir un mensaje al archivo de log del Cliente /WS, normalmente llamado ULOG.mmddyy.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newMessage, cadena de caracteres que se desea escribir al archivo de logs.

- *Parámetros de Salida*

No hay.

CLASE cTux_Service

Método **initService**

- *Objetivo*

Este método permite inicializar las variables internas de la clase, alocatear memoria y otras tareas indispensables para manejar mensajes FML32. Debe ser el primer método a llamar antes de hacer cualquier otra cosa con esta clase. Hace las veces de constructor de una clase.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newSvcName, nombre del servicio Tuxedo que se desea llamar.

- *Parámetros de Salida*

No hay.

Método **callService**

- *Objetivo*

Este método permite llamar a un servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Método **termService**

- *Objetivo*

Este método permite liberar la memoria, limpiar variables internas de la clase de tipo Servicio. Es obligatorio hacer un llamado a este método.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Método **existsField**

- *Objetivo*

Este método permite verificar la presencia de un Field en el buffer de respuesta recibido del servidor.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de la ocurrencia que se desea verificar. La primera ocurrencia es la 0, luego la 1, etc. ocurrencias $\in$ { 0,1,2,....N}

- *Parámetros de Salida*

Receive Boolean: newExists, TRUE si la ocurrencia para el field solicitado existe y FALSE cuando no existe.

Método **tranBegin**

- *Objetivo*

Este método permite iniciar una transacción desde el cliente. Sólo se debe llamar si a nivel de Tuxedo hay un Resource Manager.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

Number: newTimeout, timeout especificado en segundos.

- *Parámetros de Salida*

No hay.

Método **tranCommit**

- *Objetivo*

Este método permite hacer commit a una transacción desde el cliente. Sólo se debe llamar si a nivel de Tuxedo hay un Resource Manager y antes fue inicializada una transacción con tranBegin.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Método **tranRollback**

- *Objetivo*

Este método permite abortar una transacción desde el cliente. Sólo se debe llamar si a nivel de Tuxedo hay un Resource Manager y antes fue inicializada una transacción con tranBegin.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getMessageCode**

- *Objetivo*

Propiedad que permite obtener el código del mensaje que es enviado desde el servidor. Normalmente este mensaje lo genera el motor de Base de Datos que indica si se hizo una operación en forma exitosa o no. Este código indica niveles de errores, lo niveles definidos son 3:
58000, código de mensaje de información.
68000, código de mensaje de warning.
78000, código de mensaje de error.
Cuando el valor es 0, significa que no hay un mensaje del servidor.

- *Retorno*

Number: retorna el nivel de error correspondiente, [0, 58000, 68000, 78000].

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getMessageText**

- *Objetivo*

Propiedad que permite obtener el Texto del mensaje que es enviado desde el servidor. Normalmente este mensaje lo genera el motor de Base de Datos que indica si se hizo una operación en forma exitosa o no. Debe quedar claro que pueden existir otros mensajes que no necesariamente los genera el Motor de Base de Datos, un ejemplo de ello es cuando un servicio espera un parámetro que no le fue enviado por el cliente.
Cuando el valor es " ", significa que no hay un mensaje del servidor.

- *Retorno*

String: retorna el nivel texto que se debe desplegar en la pantalla del cliente.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getNroFilasCO**

- *Objetivo*

Propiedad que permite obtener el número de filas que vienen desde el servidor, este método sólo es útil cuando un servicio posee parámetros de tipo ColumnOutput. Si este número es mayor que 0, el cliente debiese entrar en un ciclo de N = getNroFilasCO iteraciones para rescatar cada una de las filas.

- *Retorno*

Number: retorna el número de filas de respuesta que envía el servicio, [0,1,2,3,.....]

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **setMaxFilasCO**

- *Objetivo*

Esta propiedad permite especificar el número máximo de filas que el cliente está dispuesto a recibir, posee un valor por defecto de 1000, por lo que no es necesario usar este método, salvo que se desee llamar a un servicio con parámetros de tipo ColumnOutput y que además exista una alta probabilidad de retornar una cantidad de filas mayor al defecto.

- *Retorno*

Boolean: siempre retorna TRUE.

- *Parámetros de Entrada*

Number: newMaxFilasCO, es el número máximo de filas que el cliente está dispuesto a recibir.

- *Parámetros de Salida*

No hay.

Propiedad **getMoreData**

- *Objetivo*

Esta propiedad permite verificar si hay más datos que rescatar desde el servidor Tuxedo. Sólo tiene sentido cuando el servicio que se llamó tiene parámetros de tipo ColumnOutput. Debe quedar claro que el servicio Tuxedo observa el máximo valor de filas que puede recibir el cliente y los datos que efectivamente rescata del motor de Datos o de SAP y en base a eso determina el valor de este campo. Otro aspecto importante de mencionar es que Tuxedo no guarda los datos que faltaron por enviar y que para solucionarlo hay que considerar lógica especial en el diseño de los servicios.
Por último se debe mencionar que este valor normalmente es 0, salvo que el servicio llamado posea una cantidad de filas resultantes mayor al especificado con setMaxFilasCO(...).

- *Retorno*

Number: retorna 1 si hay más filas por rescatar y 0 si no hay más filas.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **setNroFilasCI**

- *Objetivo*

Este método permite especificar el número de filas que el cliente desea enviar al servicio Tuxedo, siempre y cuando este posea parámetros de tipo ColumnInput, en caso contrario no tiene sentido utilizarlo.

- *Retorno*

Boolean: siempre retorna TRUE.

- *Parámetros de Entrada*

Number: newNroFilasCI, número de filas que el cliente desea enviar al servicio, defecto = 0.

- *Parámetros de Salida*

No hay.

Método **addFieldLong**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_LONG al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.
Number: newFieldValue, valor del campo long.

- *Parámetros de Salida*

No hay.

Método **addFieldShort**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_SHORT al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.
Number: newFieldValue, valor del campo short.

- *Parámetros de Salida*

No hay.

Método **addFieldString**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_STRING al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.
String: newFieldValue, valor del campo string.

- *Parámetros de Salida*

No hay.

Método **addFieldFloat**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_FLOAT al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.
Number: newFieldValue, valor del campo float.
Number: newDecimals, número de decimales en el número a enviar.

- *Parámetros de Salida*

No hay.

Método **addFieldDouble**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_DOUBLE al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.
Number: newFieldValue, valor del campo double.
Number: newDecimals, número de decimales en el número a enviar.

- *Parámetros de Salida*

No hay.

Método **addFieldChar**

- *Objetivo*

Este método permite agregar un campo o field de tipo FLD_CHAR al buffer que será enviado al servicio Tuxedo.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del field que se desea agregar.

String: newFieldValue, valor del campo char, debe tener largo 1.

- *Parámetros de Salida*

No hay.

Propiedad **getFieldLong**

- *Objetivo*

Este método permite obtener un campo o field de tipo FLD_LONG del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive Number: outFieldValue, valor obtenido del buffer

Propiedad **getFieldShort**

- *Objetivo*

Esta propiedad permite obtener un campo o field de tipo FLD_SHORT del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive Number: outFieldValue, valor obtenido del buffer

Propiedad **getFieldString**

- *Objetivo*

Este método permite obtener un campo o field de tipo FLD_STRING del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive String: outFieldValue, valor obtenido del buffer.

Propiedad **getFieldFloat**

- *Objetivo*

Este método permite obtener un campo o field de tipo FLD_FLOAT del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive Number: outFieldValue, valor obtenido del buffer.

Propiedad **getFieldDouble**

- *Objetivo*

Este método permite obtener un campo o field de tipo FLD_DOUBLE del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive Number: outFieldValue, valor obtenido del buffer.

Propiedad **getFieldChar**

- *Objetivo*

Esta propiedad permite obtener un campo o field de tipo FLD_CHAR del buffer que se recibió del callService.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newFieldName, nombre del campo o field que se desea rescatar.
Number: newOccurrence, número de ocurrencia que se desea rescatar.

- *Parámetros de Salida*

Receive String: outFieldValue, valor obtenido del buffer.

Propiedad **getErrorCode**

- *Objetivo*

Este método permite obtener el código de error Tuxedo.

- *Retorno*

Number: número del error Tuxedo.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Propiedad **getErrorText**

- *Objetivo*

Esta propiedad permite obtener el texto del error Tuxedo.

- *Retorno*

String: texto descriptivo del error Tuxedo.

- *Parámetros de Entrada*

No hay.

- *Parámetros de Salida*

No hay.

Método **ulog**

- *Objetivo*

Este método permite escribir un mensaje al archivo de log del Cliente /WS, normalmente llamado ULOG.mmddyy.

- *Retorno*

Boolean: retorna TRUE en éxito y FALSE cuando hay error.

- *Parámetros de Entrada*

String: newMessage, cadena de caracteres que se desea escribir al archivo de logs.

- *Parámetros de Salida*

No hay.

ANEXO No 4-3

ARCHIVO DE CONFIGURACIÓN BELLSOUTH.UBB

```
#ident "@(#) apps/simpapp/ubbsimple $Revision: 1.1 $"
#
# Archivo de configuracion TUXEDO para la aplicacion "taller" de BellSouth.
#
##########################################################################
*RESOURCES
#
#  IPCKEY Definine el ID de la seccion de memoria compartida en los
#        recursos IPC del Unix. Este ID debe ser unico en el sistema.
#
IPCKEY          140000

DOMAINID        bellsouth  # Identificacion del Dominio
MASTER          NODO1
MAXACCESSERS    1000       # Maximo de procesos accesando el BBL
MAXSERVERS      100        # Maximo de servidores en el dominio
MAXSERVICES     300        # Maximo de Servicios en esta configuracion
MAXQUEUES       100        # Maximo de Colas
MODEL           SHM        # Tipo de configuracion SHM o MP
LDBAL           Y          # "Y" indica que tuxedo debe activar el
                           #      balanceo de carga
SECURITY        NONE       # Valores posibles:
                           # NONE, APP_PW, USER_AUTH, ACL, or
                           # MANDATORY_ACL

#
# Defining IPC Limits
# -------------------
# Because most IPC and Shared Memory Bulletin Board tables are statically
# allocated for speedy processing, it is important to tune them correctly.
# If they are sized too # generously, memory and IPC resources are
# consumed to excess; if they are set too small, the process fails when
# the limits are eclipsed.
# The following tunable parameters related to IPC sizing are currently
# available in the RESOURCES section:
#
# MAXACCESSERS: the maximum number of overall processes allowed to be
#     attached to the BEA TUXEDO system at one site. It is not the sum
#     of all processes, but is equal to the number at the site that has
#     the most processes. The default is 50. (You can overwrite
#     MAXACCESSERS on a per-machine basis in the MACHINES section.)
#
# MAXSERVERS: the maximum number of server processes in the application,
#     including all the administrative servers (for example, BBL and TMS).
#     It is the sum of the server processes at all sites. The default is 50.
#
# MAXSERVICES: the maximum number of different services that can be
#     advertised in the application. It is the sum of all services in
#     the system. The default is 100. (When setting this value, consider
#     the defaults to be a quantity reserved for system resources.)
#
# The cost incurred by increasing MAXACCESSERS is one additional
# semaphore per site per accesser. There is a small fixed semaphore
# overhead for system processes in addition to that added by the
# MAXACCESSERS value. The cost of increasing MAXSERVERS and MAXSERVICES
# is a small amount of shared memory that is kept for each server,
# service, and client entry, respectively. The general idea for these
# parameters is to allow for future growth of the application. It is
# more important to scrutinize MAXACCESSERS.
#
# Note: Two additional parameters, MAXGTT and MAXCONV, affect shared memory.
#
# Characteristics of MAXACCESSERS, MAXSERVERS, and MAXSERVICES Parameters
#
# Parameter Characteristics
#
# MAXACCESSERS
#            Specifies the default maximum number of processes that
#            can have access to a bulletin board on a particular
#            processor at any one time. System administration processes,
#            such as the BBL and tmadmin, need not be accounted for in
#            this figure. This value must be greater than 0 and less than
#            32,768. If not specified, the default value is 50. The
#            *RESOURCES value for this parameter can be overridden in
```

```
#             the *MACHINES section on a per-processor basis.
#             The cost is one additional semaphore per accesser.
#
# MAXSERVERS
#             specifies the maximum number of servers to be accommodated
#             in the server table of the bulletin board. This value must
#             be greater than 0 and less than 8192. If not specified,
#             the default value is 50.
#             The cost is a small amount of shared memory.
#
# MAXSERVICES
#             Specifies the maximum total number of services to be
#             accommodated in the services table of the bulletin board.
#             This value must be greater than 0 and less than 32,768.
#             If not specified, the default value is 100.
#             The cost is a small amount of shared memory. Default is 100.
#


############################################################################
*MACHINES
DEFAULT:
            APPDIR="/be04/tuxbell/bellsouth"
            ULOGPFX="/be04/tuxbell/bellsouth/log/ULOG"
            TUXCONFIG="/be04/tuxbell/bellsouth/bellsouth_UBBBIN"
            TUXDIR="/be04/tuxedo/6.5"

jedi         LMID=NODO1             # Nombre logico de la maquina
            MAXWSCLIENTS=800       # Maximo de /WS conectadas
            CMPLIMIT="0,4096"      # nivel de compresion de mensajes


############################################################################
*GROUPS
WS          LMID=NODO1     # Nombre del grupo, LMID Nombre Logico maquina
            GRPNO=10       # Numero asignado al grupo (Arbitrario)
            OPENINFO=NONE

JOLT        LMID=NODO1
            GRPNO=15
            OPENINFO=NONE

DMADM   LMID=NODO1
            GRPNO=20
            OPENINFO=NONE

GWADM  LMID=NODO1
            GRPNO=30
            OPENINFO=NONE

gsCtaCte       LMID=NODO1
            GRPNO=100
            OPENINFO=NONE

gsLdap         LMID=NODO1
            GRPNO=110
            OPENINFO=NONE

gsCaja         LMID=NODO1
            GRPNO=120
            OPENINFO=NONE

gsClubBell     LMID=NODO1
            GRPNO=130
            OPENINFO=NONE

gsInventa      LMID=NODO1
            GRPNO=140
            OPENINFO=NONE

gsAdminUser    LMID=NODO1
            GRPNO=150
            OPENINFO=NONE

gsVenta        LMID=NODO1
            GRPNO=160
            OPENINFO=NONE

gsFideliza     LMID=NODO1
            GRPNO=170
```

```
            OPENINFO=NONE

gsManUser           LMID=NODO1
                    GRPNO=180
                    OPENINFO=NONE

gsNotaCred          LMID=NODO1
                    GRPNO=190
                    OPENINFO=NONE

gsMail              LMID=NODO1
                    GRPNO=200
                    OPENINFO=NONE

#############################################################################
#             CLOPT="-A -pL20,120:20,10 -h"
*SERVERS
DEFAULT:
            CLOPT="-A -p1,120:1,10  --"
            REPLYQ=Y
            RESTART=Y
            GRACE=3600
            MAXGEN=5
            MAX=5

#--- Domain Admin
#
DMADM         SRVGRP=DMADM
            SRVID=20
            REPLYQ=N
            SEQUENCE=920
            CLOPT="-A"

#--- Gateway Admin
#
GWADM         SRVGRP=GWADM
            SRVID=30
            REPLYQ=N
            SEQUENCE=921
            CLOPT="-A"

#--- Gateway Domain
#
GWTDOMAIN     SRVGRP=GWADM
            SRVID=40
            RQADDR=DOMQUEUE
            SEQUENCE=922
#            CLOPT="-A"

#--- Working Station Listener sin compresion de datos
#
WSL           SRVGRP=WS
            SRVID=10
            SEQUENCE=950
            GRACE=0
            MAX=1
#--- "-T 1440", timeout que desconecta clientes que llevan mas de 24 horas inactivos
            CLOPT="-A -- -n //192.168.1.217:41000 -p 41100 -P 41999 -T 1440 -m 1 -M 16 -x 50"

#--- Working Station Listener con compresion de datos
#
WSL           SRVGRP=WS
            SRVID=11
            SEQUENCE=950
            GRACE=0
            MAX=1
#--- "-T 1440", timeout que desconecta clientes que llevan mas de 24 horas inactivos
            CLOPT="-A -- -n //192.168.1.217:42000 -p 42100 -P 42999 -T 1440 -m 1 -M 10 -x 20 -c 2048"


# -- JOLT Repository Server ----------
#
JREPSVR       SRVGRP=JOLT
            SRVID=50
            SEQUENCE=960
            GRACE=0
            CLOPT="-A -- -W -P jrepository/jrepository"
#
JSL           SRVGRP=JOLT
            SRVID=60
```

```
            SEQUENCE=961
            GRACE=0
# se quita compresion -c 2048, se quita encriptacion DES 40 bits -Z 40
            CLOPT="-A -- -n //192.168.1.217:43000 -m 1 -M 10 -x 40 -T 15 -j ANY"

#--- JOLT Relay Adapter
#
JRAD      SRVGRP=JOLT
            SRVID=70
            SEQUENCE=970
            GRACE=0
            CLOPT="-A -- -c //192.168.1.217:43000 -l //192.168.1.217:44000"

#
JSL       SRVGRP=JOLT
            SRVID=80
            SEQUENCE=961
            GRACE=0
# se quita compresion -c 2048, se quita encriptacion DES 40 bits -Z 40
            CLOPT="-A -- -n //192.168.1.217:45000 -m 1 -M 10 -x 40 -T 15 -j ANY"

#--- JOLT Relay Adapter
#
JRAD      SRVGRP=JOLT
            SRVID=90
            SEQUENCE=970
            GRACE=0
            CLOPT="-A -- -c //192.168.1.217:45000 -l //192.168.1.217:46000"

#
JSL       SRVGRP=JOLT
            SRVID=100
            SEQUENCE=961
            GRACE=0
# se quita compresion -c 2048, se quita encriptacion DES 40 bits -Z 40
            CLOPT="-A -- -n //192.168.1.217:47000 -m 1 -M 10 -x 40 -T 15 -j ANY"

#--- JOLT Relay Adapter
#
JRAD      SRVGRP=JOLT
            SRVID=110
            SEQUENCE=970
            GRACE=0
            CLOPT="-A -- -c //192.168.1.217:47000 -l //192.168.1.217:48000"

#-------------------------------------------------------------------------------
JSL       SRVGRP=JOLT
            SRVID=120
            SEQUENCE=961
            GRACE=0
            CLOPT="-A -- -n //192.168.1.217:49000 -m 1 -M 10 -x 40 -T 15 -j ANY"

JRAD      SRVGRP=JOLT
            SRVID=130
            SEQUENCE=970
            GRACE=0
            CLOPT="-A -- -c //192.168.1.217:49000 -l //192.168.1.217:50000"

JSL         SRVGRP=JOLT
            SRVID=140
            SEQUENCE=961
            GRACE=0
            CLOPT="-A -- -n //192.168.1.217:51000 -m 1 -M 10 -x 40 -T 15 -j ANY"

JRAD        SRVGRP=JOLT
            SRVID=150
            SEQUENCE=970
            GRACE=0
            CLOPT="-A -- -c //192.168.1.217:51000 -l //192.168.1.217:52000"

JSL         SRVGRP=JOLT
            SRVID=160
            SEQUENCE=961
            GRACE=0
            CLOPT="-A -- -n //192.168.1.217:53000 -m 1 -M 10 -x 40 -T 15 -j ANY"

JRAD        SRVGRP=JOLT
            SRVID=170
            SEQUENCE=970
            GRACE=0
```

```
		CLOPT="-A -- -c //192.168.1.217:53000 -l //192.168.1.217:54000"
#-------------------------------------------------------------------------------------

#--- SERVIDOR Consulta Cta Cte.
#
sCtaCte
		CLOPT="-A -p1,120:1,10 -r -e log/sCtaCte.err --"
		SRVID=1000
		SEQUENCE=100
		SRVGRP=gsCtaCte
		RQADDR=qsCtaCte
		GRACE=0
		MIN=6
		MAX=15

sLdap
		CLOPT="-A -p1,120:1,10 -r -e log/sLdap.err --"
		SRVID=1010
		SEQUENCE=100
		SRVGRP=gsLdap
		RQADDR=qsLdap
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Sistema Cajas
#
sCaja
		CLOPT="-A -p1,120:1,10 -r -e log/sCaja.err --"
		SRVID=1020
		SEQUENCE=100
		SRVGRP=gsCaja
		RQADDR=qsCaja
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Sistema Club BellSouth
#
sClubBell
		CLOPT="-A -p1,120:1,10 -r -e log/sClubBell.err --"
		SRVID=1030
		SEQUENCE=100
		SRVGRP=gsClubBell
		RQADDR=qsClubBell
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Sistema Inventario
#
sInventa
		CLOPT="-A -p1,120:1,10 -r -e log/sInventa.err --"
		SRVID=1040
		SEQUENCE=100
		SRVGRP=gsInventa
		RQADDR=qsInventa
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Administracion Usuarios
#
sAdminUser
		CLOPT="-A -p1,120:1,10 -r -e log/sAdminUser.err --"
		SRVID=1050
		SEQUENCE=100
		SRVGRP=gsAdminUser
		RQADDR=qsAdminUser
#		  ENVFILE="./cfg/sAdminUser.env"
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Sistema Venta
#
sVenta
			CLOPT="-A -r -e log/sVenta.err --"
		SRVID=1060
		SEQUENCE=100
```

```
		SRVGRP=gsVenta
		RQADDR=qsVenta
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR Sistema Fidelizacion y Retencion
#
sFideliza
			CLOPT="-A -r -e log/sFideliza.err --"
		SRVID=1070
		SEQUENCE=100
		SRVGRP=gsFideliza
		RQADDR=qsFideliza
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR de Mantencion de Atributos de Usuarios
#
sManUser
		CLOPT="-A -r -e log/sManUser.err --"
		SRVID=1080
		SEQUENCE=100
		SRVGRP=gsManUser
		RQADDR=qsManUser
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR de Notas de Credito
#
sNotaCred
		CLOPT="-A -r -e log/sNotaCred.err --"
		SRVID=1090
		SEQUENCE=100
		SRVGRP=gsNotaCred
		RQADDR=qsNotaCred
		GRACE=0
		MIN=1
		MAX=5

#--- SERVIDOR de Envio de Mail
#
sMail
		CLOPT="-A -r -e log/sMail.err --"
		SRVID=1100
		SEQUENCE=100
		SRVGRP=gsMail
		RQADDR=qsMail
		GRACE=0
		MIN=1
		MAX=5

##############################################################################
#
# Lista del Servicios del Dominio
#
##############################################################################
*SERVICES

# Servicios de Consulta de Cta.Cte desde ISR++
#
c_PartidaS	LOAD=20
		PRIO=10
c_DetPartidaS	LOAD=20
		PRIO=10

# Servicios de la Aplicacion LDAP
#
w_TUX_RPC_LDAP	LOAD=20
		PRIO=10

# Servicios de la Aplicacion Caja
#
j_DocCtaCteS	LOAD=20
		PRIO=10
j_AnalizaDeudaI	LOAD=20
		PRIO=10
```

```
# Servicios de la Aplicacion Club Bellsouth
#
b_ProductoS          LOAD=20
                     PRIO=10
b_CanjeClubI         LOAD=20
                     PRIO=10

# Servicios de la Aplicacion Inventario
#
i_StockXBodegaS   LOAD=20
                     PRIO=10
i_StockProdS         LOAD=20
                     PRIO=10
i_StockPrecioS       LOAD=20
                     PRIO=10
i_RegProNoCoI        LOAD=20
                     PRIO=10
i_GrupoArtS          LOAD=20
                     PRIO=10
i_GrupoArtNoCoS   LOAD=20
                     PRIO=10
i_DescAlmacenS    LOAD=20
                     PRIO=10
i_BuscaProdS         LOAD=20
                     PRIO=10
i_BuscaProNoCoS   LOAD=20
                     PRIO=10

# Servicios de la Aplicacion Administracion de Usuarios
#
u_AtribUserCTS       LOAD=20
                     PRIO=10

# Servicios de la Aplicacion Administracion de Venta
#
v_ConceptoComS    LOAD=20
                     PRIO=10
m_ClaseDocVtaS    LOAD=20
                     PRIO=10
v_AtribUserS         LOAD=20
                     PRIO=10
v_CausalS            LOAD=20
                     PRIO=10
v_ListaDocS          LOAD=20
                     PRIO=10
v_PrecioFullS        LOAD=20
                     PRIO=10
v_VerStockS          LOAD=20
                     PRIO=10
v_DetalleDocS        LOAD=20
                     PRIO=10
v_EliminaDocD        LOAD=20
                     PRIO=10
v_BuscaCteS          LOAD=20
                     PRIO=10
v_ConcComDocS     LOAD=20
                     PRIO=10
v_CrearDoctoI        LOAD=20
                     PRIO=10
v_GrupoVendedS  LOAD=20
            PRIO=10

# Servicios de la Aplicacion Fidelizacion y Retencion
#
f_Herram_farS     LOAD=20
               PRIO=10
f_AplicaHerramI   LOAD=20
               PRIO=10
f_AnulaAplicU     LOAD=20
               PRIO=10
a_AutCallSWT      LOAD=20
               PRIO=10

# Servicios de la Aplicacion de Mantencion de Atributos de Usuarios
#
u_AtribUserU         LOAD=20
                     PRIO=10

#--- Servicios de Notas de Credito
#
```

n_CausalNCSapS	LOAD=20 PRIO=10
n_AdmEnviaMail	LOAD=20 PRIO=10
n_ObtFacturaS	LOAD=20 PRIO=10
n_ClaseDocVtaS	LOAD=20 PRIO=10
n_TipoCausalS	LOAD=20 PRIO=10
n_CausalNoCreS	LOAD=20 PRIO=10
n_CrearSolNCI	LOAD=20 PRIO=10
n_EstSolPendS	LOAD=20 PRIO=10

ANEXO No 4-4

ARCHIVO DE CONFIGURACIÓN BELLSOUTH.DOM

```
*DM_LOCAL_DOMAINS
ldom1                   GWGRP=GWADM
                        TYPE=TDOMAIN
                        DOMAINID="bellsouth_j"
#                       CONNECTION_POLICY=ON_STARTUP

*DM_REMOTE_DOMAINS
rdom1                   TYPE=TDOMAIN
                        DOMAINID="elink_t"

rdom2                   TYPE=TDOMAIN
                        DOMAINID="outside_dom"

*DM_TDOMAIN
ldom1                   NWADDR="//jedi:9001"
rdom1                   NWADDR="//tarkin:9001"
rdom2                   NWADDR="//organa:9001"

*DM_LOCAL_SERVICES
#
# Servicios de Consulta de Cta.Cte desde ISR++
#
c_PartidaS                      LDOM=ldom1
c_DetPartidaS                   LDOM=ldom1
#
# Servicios de Cajas
#
j_DocCtaCteS                    LDOM=ldom1
j_AnalizaDeudaI                 LDOM=ldom1
#
# Servicios de ClubBellSouth
#
b_ProductoS                     LDOM=ldom1
b_CanjeClubI                    LDOM=ldom1
#
# Servicios de la Aplicacion Inventario
#
i_StockXBodegaS                 LDOM=ldom1
#
# Servicios de la Aplicacion Administracion de Usuarios
#
u_AtribUserCTS                  LDOM=ldom1

*DM_REMOTE_SERVICES
Tx_cPartidas            LDOM=ldom1      RDOM=rdom1
Tx_cPartAbierta         LDOM=ldom1      RDOM=rdom1
Tx_cDetPartida          LDOM=ldom1      RDOM=rdom1
Tx_jDocCtaCte           LDOM=ldom1      RDOM=rdom1
Tx_StockProduct         LDOM=ldom1      RDOM=rdom1
Tx_GenPedidoVta         LDOM=ldom1      RDOM=rdom1
Tx_AnulaPedido          LDOM=ldom1      RDOM=rdom1
Tx_ConsStockCo          LDOM=ldom1      RDOM=rdom1

# Servicios de la Aplicacion Inventario
#
Tx_iBuscaProdS      LDOM=ldom1      RDOM=rdom1
Tx_vCausalS         LDOM=ldom1      RDOM=rdom1
Tx_vPedidoVtaH      LDOM=ldom1      RDOM=rdom1
Tx_vPedidoVtaI      LDOM=ldom1      RDOM=rdom1
Tx_vPedidoVtaK      LDOM=ldom1      RDOM=rdom1
Tx_vPedidoVtaL      LDOM=ldom1      RDOM=rdom1
Tx_vListaDocS       LDOM=ldom1      RDOM=rdom1
Tx_vDocVtaOrg       LDOM=ldom1      RDOM=rdom1
Tx_mClaseDocVta     LDOM=ldom1      RDOM=rdom1
Tx_vPrecioFullS     LDOM=ldom1      RDOM=rdom1
Tx_vDescAlmacen     LDOM=ldom1      RDOM=rdom1
Tx_vDetalleDocS     LDOM=ldom1      RDOM=rdom1
Tx_iBusProNoCoS     LDOM=ldom1      RDOM=rdom1
Tx_iRegProNoCoS     LDOM=ldom1      RDOM=rdom1
Tx_vBuscaCteS       LDOM=ldom1      RDOM=rdom1
Tx_vVerStockS       LDOM=ldom1      RDOM=rdom1

# Servicios de la Aplicacion Ventas e Inventario
#
Tx_vGrupoVendeS     LDOM=ldom1      RDOM=rdom1
```

Tx_uAtrUserSapS LDOM=ldom1 RDOM=rdom1
Tx_nObtFacturaS LDOM=ldom1 RDOM=rdom1
Tx_nLiberaSolU LDOM=ldom1 RDOM=rdom1

ANEXO No 4-5

ARCHIVO DE CONFIGURACIÓN ELINK.UBB

```
#
# Archivo de configuracion TUXEDO para la aplicacion "elink" de BellSouth.
#

##########################################################################
*RESOURCES
#
#  IPCKEY Definine el ID de la seccion de memoria compartida en los
#         recursos IPC del Unix. Este ID debe ser unico en el sistema.
#
IPCKEY          140000

DOMAINID        "elink" # Identificacion del Dominio.
MASTER          NODO1
MAXACCESSERS    500         # Maximo de procesos accesando el BBL
MAXSERVERS      100         # Maximo de servidores en el dominio
MAXSERVICES     300         # Maximo de Servicios en esta configuracion
MODEL           SHM         # Tipo de configuracion SHM o MP
LDBAL           Y           # "Y" indica que tuxedo debe activar el
                            #      balanceo de carga
SECURITY        NONE        # Valores posibles:
                            # NONE, APP_PW, USER_AUTH, ACL, or
                            # MANDATORY_ACL

##########################################################################
*MACHINES
DEFAULT:
        APPDIR="/home/mis/tuxsap/elink"
        ULOGPFX="/home/mis/tuxsap/elink/log/ULOG"
        TUXCONFIG="/home/mis/tuxsap/elink/elink_UBBBIN"
        TUXDIR="/home/mis/tuxedo/6.5"

tarkin      LMID=NODO1          # Nombre logico de la maquina
        MAXWSCLIENTS=50         # Maximo de /WS conectadas
        CMPLIMIT="0,4096"       # nivel de compresion de mensajes

##########################################################################
*GROUPS
DEFAULT:        LMID=NODO1
                OPENINFO=NONE

DMGRP           GRPNO=10
GWGRP1          GRPNO=20

CR3GRP          GRPNO=50

GRP_ISR                 GRPNO=100
GRP_CAJAS               GRPNO=200
GRP_CLUB_BELL           GRPNO=300
GRP_VTAS_INV            GRPNO=400
GRP_VTAS_INV_2          GRPNO=500
GRP_ADMIN_USERS                 GRPNO=600
GRP_NOTAS_CREDITO       GRPNO=700

##########################################################################
*SERVERS
DEFAULT:        RESTART=Y
        GRACE=0
        MAXGEN=10

#--- Elink server para las RFC's de Cuenta Corriente ISR++

cr3rfcin        SRVID=10
        SRVGRP=GRP_ISR
        MIN=2
        MAX=10
        RQADDR="QUEUE_ctacte_isr"
        REPLYQ=Y
        CLOPT="-r -e log/cr3rfcin_ctacte_isr.err -o log/cr3rfcin_ctacte_isr.out -- -i cr3rfcin -e
cfg/cr3rfcin_ctacte_isr.env"

#--- Elink server para las RFC's de Cuenta Corriente Cajas

cr3rfcin        SRVID=20
```

```
            SRVGRP=GRP_CAJAS
            MIN=4
            MAX=10
            RQADDR="QUEUE_ctacte_cajas"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_ctacte_cajas.err -o log/cr3rfcin_ctacte_cajas.out -- -i cr3rfcin -e
cfg/cr3rfcin_ctacte_cajas.env"

#--- Elink server para las RFC's de Club BellSouth

cr3rfcin      SRVID=30
            SRVGRP=GRP_CLUB_BELL
            MIN=4
            MAX=10
            RQADDR="QUEUE_club_bell"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_club_bell.err -o log/cr3rfcin_club_bell.out -- -i cr3rfcin -e cfg/cr3rfcin_club_bell.env"

#--- Elink server para las RFC's de Ventas e Inventario

cr3rfcin      SRVID=40
            SRVGRP=GRP_VTAS_INV
            MIN=4
            MAX=10
            RQADDR="QUEUE_vtas_inv"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_vtas_inv.err -o log/cr3rfcin_vtas_inv.out -- -i cr3rfcin -e cfg/cr3rfcin_vtas_e_inv.env"

cr3rfcin      SRVID=50
            SRVGRP=GRP_VTAS_INV_2
            MIN=4
            MAX=10
            RQADDR="QUEUE_vtas_inv_2"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_vtas_inv_2.err -o log/cr3rfcin_vtas_inv_2.out -- -i cr3rfcin -e
cfg/cr3rfcin_vtas_e_inv_2.env"

#--- Elink server para las RFC's de Administracion de Usuarios

cr3rfcin      SRVID=60
            SRVGRP=GRP_ADMIN_USERS
            MIN=4
            MAX=10
            RQADDR="QUEUE_admin_users"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_admin_users.err -o log/cr3rfcin_admin_users.out -- -i cr3rfcin -e
cfg/cr3rfcin_admin_users.env"

#--- Elink server para las RFC's de Notas de Credito

cr3rfcin      SRVID=70
            SRVGRP=GRP_NOTAS_CREDITO
            MIN=4
            MAX=10
            RQADDR="QUEUE_notas_credito"
            REPLYQ=Y
            CLOPT="-r -e log/cr3rfcin_notas_credito.err -o log/cr3rfcin_notas_credito.out -- -i cr3rfcin -e
cfg/cr3rfcin_notas_credito.env"

#--- Elink server para las RFC's que desde SAP llaman servicios Tuxedo (ejemplo)

cr3rfcout     SRVID=1000
            SRVGRP=CR3GRP
            CLOPT="-r -e log/cr3rfcout.err -o log/cr3rfcout.out -s CR3_RFC_OUT -- -i cr3rfcout -e cfg/cr3rfcout.env"

#--- Domain Admin
#
DMADM         SRVGRP=DMGRP       SRVID=100

#--- Gateway Admin
#
GWADM         SRVGRP=GWGRP1    SRVID=200

#--- Gateway Domain
#
GWTDOMAIN     SRVGRP=GWGRP1              SRVID=300

##############################################################################
```

```
#
# Lista del Servicios del Dominio
#
################################################################################
*SERVICES

# Servicios de la Aplicacion
#
CR3_RFC_IN
CR3_RFC_OUT

#--- Servicios Para Cta Cte ISR++

Tx_cPartidas            SVCTIMEOUT=60
Tx_cPartAbierta         SVCTIMEOUT=60
Tx_cDetPartida          SVCTIMEOUT=60

#--- Servicios Para Cta Cte Cajas

Tx_jDocCtaCte           SVCTIMEOUT=60

#--- Servicios Para Club BellSouth

Tx_StockProd            SVCTIMEOUT=60
Tx_GenPedidoVta         SVCTIMEOUT=60
Tx_AnulaPedido          SVCTIMEOUT=60
Tx_ConsStockCo          SVCTIMEOUT=60

#--- Servicios Para Ventas e Inventario

Tx_iBuscaProdS          SVCTIMEOUT=60
Tx_vCausalS             SVCTIMEOUT=60
Tx_vPedidoVtaH          SVCTIMEOUT=60
Tx_vPedidoVtaI          SVCTIMEOUT=60
Tx_vPedidoVtaK          SVCTIMEOUT=60
Tx_vPedidoVtaL          SVCTIMEOUT=60
Tx_vListaDocS           SVCTIMEOUT=60
Tx_vDocVtaOrg           SVCTIMEOUT=60
Tx_mClaseDocVta         SVCTIMEOUT=60
Tx_StockProduct         SVCTIMEOUT=60
Tx_vPrecioFullS         SVCTIMEOUT=60
Tx_vDescAlmacen         SVCTIMEOUT=60
Tx_vDetalleDocS         SVCTIMEOUT=60
Tx_iBusProNoCoS         SVCTIMEOUT=60
Tx_iRegProNoCoS         SVCTIMEOUT=60
Tx_vBuscaCteS           SVCTIMEOUT=60
Tx_vVerStockS           SVCTIMEOUT=60
Tx_vGrupoVendeS         SVCTIMEOUT=60
```

ANEXO No 4-6

ARCHIVO DE CONFIGURACIÓN ELINK.DOM

```
*DM_LOCAL_DOMAINS
ldom1           GWGRP=GWGRP1
                TYPE=TDOMAIN
                DOMAINID="elink_t"

*DM_REMOTE_DOMAINS
rdom1           TYPE=TDOMAIN
                DOMAINID="bellsouth_j"

rdom2           TYPE=TDOMAIN
                DOMAINID="bellsouth_w"

*DM_TDOMAIN
ldom1           NWADDR="//tarkin:9001"
rdom1           NWADDR="//jedi:9001"
rdom2           NWADDR="//wedge:9001"

*DM_LOCAL_SERVICES
Tx_cPartidas            LDOM=ldom1
Tx_cPartAbierta         LDOM=ldom1
Tx_cDetPartida      LDOM=ldom1
Tx_StockProduct         LDOM=ldom1
Tx_GenPedidoVta         LDOM=ldom1
Tx_AnulaPedido          LDOM=ldom1
Tx_ConsStockCo          LDOM=ldom1

Tx_jDocCtaCte       LDOM=ldom1

Tx_iBuscaProdS          LDOM=ldom1
Tx_vCausalS             LDOM=ldom1
Tx_vPedidoVtaH          LDOM=ldom1
Tx_vPedidoVtaI          LDOM=ldom1
Tx_vPedidoVtaK          LDOM=ldom1
Tx_vPedidoVtaL          LDOM=ldom1
Tx_vListaDocS           LDOM=ldom1
Tx_vDocVtaOrg           LDOM=ldom1
Tx_mClaseDocVta         LDOM=ldom1
Tx_vPrecioFullS         LDOM=ldom1
Tx_vDescAlmacen         LDOM=ldom1
Tx_vDetalleDocS         LDOM=ldom1
Tx_iBusProNoCoS         LDOM=ldom1
Tx_iRegProNoCoS         LDOM=ldom1
Tx_vBuscaCteS           LDOM=ldom1
Tx_vVerStockS           LDOM=ldom1

Tx_vGrupoVendeS     LDOM=ldom1

Tx_uAtrUserSapS         LDOM=ldom1
Tx_nObtFacturaS         LDOM=ldom1
Tx_nLiberaSolU          LDOM=ldom1


*DM_REMOTE_SERVICES
```

ANEXO No 4-7

SERVIDOR DE CUENTA CORRIENTE

```
/*
**  Archivo: scctacte.pc
**
*** Implementa los servicios Tuxedo  de consulta cta cte.
**
**  Nombre del servidor Tuxedo: scctacte
**  Nombre de los Servicios   :
**                          c_PartidaS
**                          c_DetPartidaS
**
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sqlda.h>
#include <sqlcpr.h>
#include <sqlca.h>

#include <atmi.h>
#include <fml32.h>
#include <fml1632.h>
#include <userlog.h>

#include <tux_utils.h>
#include <TblGlobal.fld.h>

/*
** Define largos para las variables del SQL.
** El primer valor corresponde al largo en la BD
** y el segundo valor permite agregar '\0' para
** las cadenas.
*/
#define    LEN_FLD_NROCLIENTE              9  + 1
#define    LEN_FLD_NROCUENTA               9  + 1
#define    LEN_FLD_SAPID                  16  + 1
#define    LEN_FLD_CODEMPRESAFACT          4   + 1
#define    LEN_FLD_SAPSALDO               20  + 1
#define    LEN_FLD_SAPIMPORTE             20  + 1
#define    LEN_FLD_FOLIOINTERNOSAP        10  + 1

#define    LEN_FECHAENTRADADOCCONT   LEN_FECHAS
#define    LEN_FECHACONTABILIZACION  LEN_FECHAS

/*
** Nombre de los servicios Tuxedo que se llaman a traves de eLink.
** En el fondo estos son RFCs de SAP
*/
#define    TODAS_LAS_PARTIDAS      "Tx_cPartidas"     /* Nombre Svc. Tuxedo para Todas las Partidas     */
#define    PARTIDAS_ABIERTAS       "Tx_cPartAbierta"  /* Nombre Svc. Tuxedo para Partidas  Abiertas     */
#define    DETALLE_FACTURA         "Tx_cDetPartida"   /* Nombre Svc. Tuxedo para el Detalle de Factura  */

/*
** Typedef de arrays.
*/

EXEC SQL BEGIN DECLARE SECTION;


      /*
      ** VARIABLES NO USADAS POR EL MOMENTO, SOLO DE REFERENCIA
      */
      short          o_Fld_NroFilas;
      VARCHAR        loc_MessageSP[LEN_MESSAGE];
      char           GV_ServerName[78+1];

      /*
      ** VARIABLES AUXILIARES O LOCALES
      */
      VARCHAR        loc_fld_SapId[LEN_FLD_SAPID];
```

```
        /*
        ** PARAMETROS DE ENTRADA AL SERVICIO TUXEDO
        */
        char            i_fld_CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];
        char            i_fld_FolioInternoSap[LEN_FLD_FOLIOINTERNOSAP];
        char            i_fld_FechaContabilizacion[LEN_FECHACONTABILIZACION];
        char            i_fld_TipoConsulta;
        char            i_fld_NroCliente[LEN_FLD_NROCLIENTE];
        char            i_fld_NroCuenta[LEN_FLD_NROCUENTA];

        /*
        ** PARAMETROS DE SALIDA AL SERVICIO TUXEDO
        */

EXEC SQL END DECLARE SECTION;


/*
** PROTOTIPOS PARA LAS LLAMADAS A LOS RFCs
*/
TYPE_STATUS call_RFC_Partidas( char TipoConsultaIN, char *CodEmpresaFactIN, char *SapIdIN,
TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOut);
TYPE_STATUS call_RFC_Detalle(  char *CodEmpresaFactIN, char *FolioInternoSapIN, char *FechaContabilizacionIN ,
TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOut);

#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
c_PartidaS(TPSVCINFO *rqst)
#else
c_PartidaS(rqst)
TPSVCINFO *rqst;
#endif
{
        TYPE_STATUS          ora_status;
        char                 ora_message[LEN_MESSAGE];
        long                 SQLCODE=0;

        short                nro_filas;
        TYPE_STATUS          ret;
        TYPE_EXISTS          field_exists;

        FBFR                 *fbfr=(FBFR *)0;
        FBFR                 *fbfr_aux=(FBFR *)0;
        FLDOCC               ocurr;

        TYPE_HEADERMESSAGE    hdr;
        char                 priv_fld_StatusUser;
        char                 msg[LEN_MESSAGE];


        /*********************************************************
          PASE LO QUE PASE SIGA, NO SALIR CON EXIT O ALGO ASI
        *********************************************************/
        EXEC SQL WHENEVER SQLERROR  CONTINUE;
        EXEC SQL WHENEVER NOT FOUND CONTINUE;


        /*********************************************************
             OBTIENE BUFFER FML DE LOS DATOS DE ENTRADA
        *********************************************************/
        fbfr = (FBFR *)rqst->data;


        /*********************************************************
           INICIALIZA EL HEADER DEL MENSAJE TUXEDO
           CON VALORES POR DEFECTO
        *********************************************************/
        ret = tux_initHeader(&hdr, rqst->name );


        /*********************************************************
           CHEQUEO DE PRIVILEGIOS

        *********************************************************/
#if defined(WITH_PRIVI)
        priv_fld_StatusUser = ' ';
```

```
    ret = tux_CheckPrivi(GV_ServerName, rqst, &hdr, &priv_fld_StatusUser );
    if(ret == STATUS_FAIL)
    {
        userlog("[%s]: Usuario sin acceso al Servicio[%s] Causa[%s]Status[%c]" , rqst->name , rqst->name,  \
                                hdr.o_fld_hdr_MsgText,priv_fld_StatusUser);
        sprintf(msg,"Usuario sin acceso al Servicio[%s] Causa[%s]"   , rqst->name, hdr.o_fld_hdr_MsgText);
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
        goto _error;
    }
#endif

    /*******************************************************
       OBTIENE EL HEADER DEL MENSAJE TUXEDO
       QUE ENVIO EL CLIENTE
    *******************************************************/
    if (tux_getHeader(rqst, &hdr ) == STATUS_FAIL )
    {
        userlog("[%s]: Fallo llamada a tux_getHeader" , rqst->name );
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Error al obtener Header del mensaje de entrada" , &hdr );
        goto _error;
    }


    /*******************************************************
       OBTIENE LOS PARAMETROS DE ENTRADA
       PROPIOS DEL SERVICIO.
    *******************************************************/
    i_fld_CodEmpresaFact[0]= '\0';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_CodEmpresaFact, ocurr, i_fld_CodEmpresaFact, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_CodEmpresaFact  no fue recibido en msg. de entrada", \
                                                                    rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_CodEmpresaFact no fue recibido en \
                                                                        msg. de entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_CodEmpresaFact)=[%d][%s]", rqst->name , Ferror,\
                                                                        Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibido fld_CodEmpresaFact [%s]\n", rqst->name , i_fld_CodEmpresaFact );


    i_fld_TipoConsulta = ' ';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_TipoConsulta, ocurr, (char *)&i_fld_TipoConsulta, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_TipoConsulta  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_TipoConsulta  no fue recibido en msg. \
                                                        de entrada", &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_NroCliente)=[%d][%s]", rqst->name , Ferror,\
                                        Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    if ( (i_fld_TipoConsulta != 'A') && (i_fld_TipoConsulta != 'T'))
    {
            userlog("[%s]: Advertencia, se recibio fld_TipoConsulta [%c] fuera de rango [A,T]", rqst->name,\
                                                        i_fld_TipoConsulta );
        i_fld_TipoConsulta = 'T';
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibido fld_TipoConsulta [%c]", rqst->name , i_fld_TipoConsulta );
```

```
i_fld_NroCliente[0]= '\0';
ocurr = 0;
if (( tux_getField(fbfr, fld_NroCliente, ocurr, i_fld_NroCliente, &field_exists) ) == STATUS_FAIL)
{
      if(field_exists == FIELD_NOT_EXISTS)
      {
            userlog("[%s]:ADVERTENCIA: campo fld_NroCliente  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_NroCliente no fue recibido en msg. De\
                                                                                                  entrada" , &hdr );
      }
      else
      {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_NroCliente)=[%d][%s]", rqst->name ,\
                                                                                  Ferror, Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
      }
      goto _error;
}
else if(hdr.WithDebug)
      userlog("[%s]:Se recibido fld_NroCliente [%s]\n", rqst->name , i_fld_NroCliente );



i_fld_NroCuenta[0] = '\0';
ocurr = 0;
if ((tux_getField(fbfr, fld_NroCuenta, ocurr, i_fld_NroCuenta, &field_exists) ) == STATUS_FAIL)
{
      if(field_exists == FIELD_NOT_EXISTS)
      {
            userlog("[%s]:ADVERTENCIA: campo fld_NroCuenta  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_NroCuenta no fue recibido en msg. De\
                                                                                              entrada",&hdr);
      }
      else
      {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_NroCuenta)=[%d][%s]", rqst->name , Ferror,\
                                                                                  Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
      }
      goto _error;
}
else if(hdr.WithDebug)
      userlog("[%s]:Se recibido fld_NroCuenta [%s]\n", rqst->name , i_fld_NroCuenta );

/*********************************************************
  Rescate del SAPID del Cliente/Cuenta a
  traves de un Procedimeinto Almacenado
*********************************************************/
EXEC SQL EXECUTE
      BEGIN c_SapIdS_Pkg.c_SapIdS
                  (:loc_MessageSP,
                   :i_fld_NroCliente,
                   :i_fld_NroCuenta,
                   :loc_fld_SapId);
      END;
END-EXEC;

ret = ora_checkStatusSql((char *)loc_MessageSP.arr , &ora_status , ora_message);
if(ora_status != ORA_STATUS_SUCCESS)
{
            /*
            ** Aqui podria ponerse logica para saber cuando hay mensajes de tipo information
            */
      ret = tux_setMsgHeader( ora_status , ora_message , &hdr );
      goto  _error;
}

if( call_RFC_Partidas( i_fld_TipoConsulta, i_fld_CodEmpresaFact, (char *)loc_fld_SapId.arr, &hdr , \
                                                      (FBFR **)&fbfr_aux , &nro_filas) == STATUS_FAIL )
{
      userlog("[%s]: Error en llamada a call_RFC_Partidas", rqst->name);
      goto  _error;
}
```

```
	hdr.o_fld_hdr_NroFilasCO = nro_filas;
	if(hdr.WithDebug)
		userlog("[%s]: Nro de filas recibidas[%d]",  rqst->name, nro_filas );


	goto _exit;


_error:
	ret = tux_setDataError(rqst, hdr);
	if(fbfr_aux) tux_tpfreeFml(fbfr_aux);
	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);

_exit:
	ret = tux_setData(rqst, fbfr_aux,  hdr);
	if(fbfr_aux) tux_tpfreeFml(fbfr_aux);

	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);
}
```

```
#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
c_DetPartidaS(TPSVCINFO *rqst)
#else
c_DetPartidaS(rqst)
TPSVCINFO *rqst;
#endif
{


	short		nro_filas;
	TYPE_STATUS	ret;
	TYPE_EXISTS	field_exists;

	FBFR		*fbfr=(FBFR *)0, *fbfr_aux=(FBFR *)0;
	FLDOCC		ocurr;

	TYPE_HEADERMESSAGE  hdr;
	char		priv_fld_StatusUser;
	char		msg[LEN_MESSAGE];

	/*********************************************************
	     OBTIENE BUFFER FML DE LOS DATOS DE ENTRADA
	*********************************************************/
	fbfr = (FBFR *)rqst->data;


	/*********************************************************
	   INICIALIZA EL HEADER DEL MENSAJE TUXEDO
	   CON VALORES POR DEFECTO
	*********************************************************/
	ret = tux_initHeader(&hdr, rqst->name );


	/*********************************************************
	   CHEQUEO DE PRIVILEGIOS

	*********************************************************/
#if defined(WITH_PRIVI)
	priv_fld_StatusUser = ' ';
	ret = tux_CheckPrivi(GV_ServerName, rqst, &hdr, &priv_fld_StatusUser );
	if(ret == STATUS_FAIL)
	{
		userlog("[%s]: Usuario sin acceso al Servicio[%s] Causa[%s]Status[%c]" , rqst->name , rqst->name,\
                                                                        hdr.o_fld_hdr_MsgText,priv_fld_StatusUser);
		sprintf(msg,"Usuario sin acceso al Servicio[%s] Causa[%s]"   , rqst->name, hdr.o_fld_hdr_MsgText);
		ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
		goto _error;
	}
#endif

	/*********************************************************
	   OBTIENE EL HEADER DEL MENSAJE TUXEDO
	*********************************************************/
	if (tux_getHeader(rqst, &hdr ) == STATUS_FAIL )
	{
		userlog("[%s]: Fallo llamada a tux_getHeader" , rqst->name );
		ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Error al obtener Header del mensaje de entrada" , &hdr );
		goto _error;
	}


	/*********************************************************
	   OBTIENE LOS PARAMETROS DE ENTRADA
	   PROPIOS DEL SERVICIO.
	*********************************************************/
	i_fld_CodEmpresaFact[0]= '\0';	/* LEN_FLD_CODEMPRESAFACT  */
	ocurr = 0;
	if (( tux_getField(fbfr, fld_CodEmpresaFact, ocurr, i_fld_CodEmpresaFact, &field_exists) ) == STATUS_FAIL)
	{
		if(field_exists == FIELD_NOT_EXISTS)
		{
			userlog("[%s]:ADVERTENCIA: campo fld_CodEmpresaFact  no fue recibido en msg. de entrada",\
                                                                                rqst->name );
			ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_CodEmpresaFact no fue \
```

```
                                                    recibido en msg. de entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_CodEmpresaFact)=[%d][%s]", rqst->name , Ferror,\
                                                                                                    Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
}
else if(hdr.WithDebug)
        userlog("[%s]:Se recibido fld_CodEmpresaFact [%s]\n", rqst->name , i_fld_CodEmpresaFact );


i_fld_FolioInternoSap[0] = '\0';
ocurr = 0;
if (( tux_getField(fbfr, fld_FolioInternoSap, ocurr, i_fld_FolioInternoSap, &field_exists) ) == STATUS_FAIL)
{
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_FolioInternoSap  no fue recibido en msg. de entrada",\
                                                                                        rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_FolioInternoSap  no fue recibido \
                                                                                    en msg. de entrada", &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_FolioInternoSap)=[%d][%s]", rqst->name , Ferror,\
                                                                                                    Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
}
else if(hdr.WithDebug)
        userlog("[%s]:Se recibido fld_FolioInternoSap [%s]", rqst->name , i_fld_FolioInternoSap );


i_fld_FechaContabilizacion[0]= '\0';
ocurr = 0;
if (( tux_getField(fbfr, fld_FechaContabilizacion, ocurr, i_fld_FechaContabilizacion, &field_exists) )
                                                                                        == STATUS_FAIL)
{
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_FechaContabilizacion  no fue recibido en msg. de entrada",\
                                                                                                    rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_FechaContabilizacion no recibido en \
                                                    msg. entrada" , &hdr );
        }
        else
        {
           userlog("[%s]: ERROR en llamada a tux_getField(fld_FechaContabilizacion)=[%d][%s]", rqst->name, Ferror,\
                                                                                                    Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
}
else if(hdr.WithDebug)
        userlog("[%s]:Se recibido fld_FechaContabilizacion [%s]\n", rqst->name , i_fld_FechaContabilizacion );


/*********************************************************
  En este servicio no se requiere acceso a
  la base de datos, por lo que solo se procedera a
  llamar al RFC en forma directa.
*********************************************************/
if( call_RFC_Detalle( i_fld_CodEmpresaFact, i_fld_FolioInternoSap, i_fld_FechaContabilizacion, &hdr , \
                                                    (FBFR **)&fbfr_aux, &nro_filas) == STATUS_FAIL )
{
        userlog("[%s]: Error en llamada a call_RFC_Detalle", rqst->name);
        goto  _error;
}
```

```
	hdr.o_fld_hdr_NroFilasCO = nro_filas;
	if(hdr.WithDebug)
		userlog("[%s]: Nro de filas recibidas[%d]",  rqst->name, nro_filas );


	goto _exit;

_error:
	ret = tux_setDataError(rqst, hdr);
	if( fbfr_aux ) tux_tpfreeFml(fbfr_aux);
	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);

_exit:
	ret = tux_setData(rqst, fbfr_aux,  hdr);
	if( fbfr_aux ) tux_tpfreeFml(fbfr_aux);
	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);
}




/*
** Objetivo:
**            LLama al RFC que corresponda segun valor TipoConsultaIN
**
** Parametros de entrada:
**
** Parametros de salida :
**
*/
TYPE_STATUS call_RFC_Partidas ( char TipoConsultaIN, char *CodEmpresaFactIN, char *SapIdIN, \
                                        TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOUT)
{
	long          def_ret     = STATUS_FAIL;
	char          fechaAux[LEN_FECHAS];
	char          fechaActual[LEN_FECHAS];
	char         *fechaDesde  = FECHA_DEF_YYYYMMDD;
	char          serviceName[16], msg[LEN_MESSAGE];
	long          fbfr_size   = 1024*4, real_size;
	long          rcvlen, ret;
	short         idx;
	FLDOCC        nroFilas;
	FLDOCC        ocurr;
	FBFR          *fbfr_send;
	FBFR          *fbfr_recv;

	char          _fld_IndEstadistica;
	char          _fld_IndicadorDH;
	char          _fld_SapDebeHaber[24+1];
	double        _fld_Debe, _fld_Haber, _fld_Saldo;
	char          _fld_SapSaldo[LEN_FLD_SAPSALDO];
	TYPE_EXISTS    field_exists;


	static   FLDID
		field_arr[]={
				fld_FolioInternoSap
				,fld_IndOperacionCtaEsp
				,fld_FechaEntradaDocCont
				,fld_FechaEmision
				,fld_DocRef
				,fld_TipoDoc
				,fld_GlosaTipoDoc
				,fld_IndicadorDH
				,fld_GlosaDetalle
				,fld_FechaVctoPago
				,fld_FechaContabilizacion
				,fld_SapDebeHaber     /* Corresponde al campo AMOUNT                    */
				,fld_FolioDocumento   /* Corresponde al campo ALLOC_NMBR                */
				,fld_SapSaldo         /* Corresponde al campo SALDO en formato char(20)   */
				,BADFLDID
```

```
                        };



*nroFilasOUT = 0;
ret = tux_getCurrDate(fechaActual, FORMAT_YYYYMMDD);


/*
** Alocatea espacio e inicializa buffers FML.
*/
if( (fbfr_send=(FBFR *)tpalloc(FMLTYPE, NULL, fbfr_size)) == NULL)
{
      userlog("call_RFC_Partidas: Error en tpalloc [%d][%s]", tperrno, tpstrerror(tperrno));
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , tpstrerror(tperrno) , hdrIN );
      return(def_ret);
}


/*
** Agrega los parametros de entrada para llamar al RFC como un
** servicio mas de Tuxedo.
*/
if ((tux_addField(fbfr_send, fld_SapId , SapIdIN ) == STATUS_FAIL))
{
      userlog("call_RFC_Partidas: error en llamada a tux_addField(fld_SapId) ERROR[%s]",Fstrerror(Ferror));
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
      tpfree((char *)fbfr_send);
      return(def_ret);
}

if ((tux_addField(fbfr_send, fld_CodEmpresaFact , CodEmpresaFactIN ) == STATUS_FAIL))
{
      userlog("call_RFC_Partidas: error en llamada a tux_addField(fld_CodEmpresaFact) \
                                                                        ERROR[%s]",Fstrerror(Ferror));
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
      tpfree((char *)fbfr_send);
      return(def_ret);
}

if (TipoConsultaIN == 'T' )
{
      if ((tux_addField(fbfr_send, fld_SapFechaDesde , fechaDesde ) == STATUS_FAIL))
      {
            userlog("call_RFC_Partidas: error en llamada a \
                                          tux_addField(fld_SapFechaDesde) ERROR[%s]",Fstrerror(Ferror));
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
            tpfree((char *)fbfr_send);
            return(def_ret);
      }
      strcpy(serviceName,TODAS_LAS_PARTIDAS );
}
else
      strcpy(serviceName, PARTIDAS_ABIERTAS );

if ((tux_addField(fbfr_send, fld_SapFechaHasta , fechaActual ) == STATUS_FAIL))
{
     userlog("call_RFC_Partidas: error en llamada a tux_addField(fld_SapFechaHasta/Actual)\
                                                                        ERROR[%s]",Fstrerror(Ferror));
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
      tpfree((char *)fbfr_send);
      return(def_ret);
}


/* Este campo salio a ultima hora y es un indicador de estadistica, se especifica
** que se mandara como espacio en blanco
*/
_fld_IndEstadistica = ' ';
if ((tux_addField(fbfr_send, fld_IndEstadistica , (char *)&_fld_IndEstadistica ) == STATUS_FAIL))
{
   userlog("call_RFC_Partidas: error en llamada a tux_addField(fld_IndEstadistica) ERROR[%s]",Fstrerror(Ferror));
   ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
   tpfree((char *)fbfr_send);
   return(def_ret);
}
```

```
/*
** Alocatea e inicializa buffer para recibir los datos
*/
if( (fbfr_recv=(FBFR *)tpalloc(FMLTYPE, NULL, fbfr_size)) == NULL)
{
      userlog("call_RFC_Partidas: Error en tpalloc [%d][%s]", tperrno, tpstrerror(tperrno));
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , tpstrerror(tperrno) , hdrIN );
      tpfree((char *)fbfr_send);
      return(def_ret);
}
rcvlen = fbfr_size;


/*
** Tpcall al servicio via elink
*/
if ( tpcall(serviceName, (char *)fbfr_send, 0L, (char **)&fbfr_recv, &rcvlen, (long)0) == -1)
{
     sprintf(msg, "Error al llamar a SAP via tpcall(%s) , Error del tuxedo[%d][%s]" ,serviceName ,tperrno,\
                                                                                    tpstrerror(tperrno));
     ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , hdrIN );
      userlog("call_RFC_Partidas: Error en tpcall [%d][%s]", tperrno, tpstrerror(tperrno));
      tpfree((char *)fbfr_send);
      tpfree((char *)fbfr_recv);
      return(def_ret);
}
tpfree((char *)fbfr_send);




/*
** Procesamiento de los datos de respuesta desde el RFC de SAP.
*/
nroFilas = Foccur(fbfr_recv, field_arr[0]);

if(nroFilas == 0)
{
      if(hdrIN->WithDebug)
            userlog("call_RFC_Partidas: No hay datos asociados a la consulta. \
            TipoConsulta[%c]CodEmpresaFact[%s]SapId[%s]", \
            TipoConsultaIN, \
            CodEmpresaFactIN,SapIdIN);
      ret = tux_setMsgHeader( HDR_LEVEL_INFORMATION , "No hay datos asociados a la consulta." , hdrIN );
      tpfree((char *)fbfr_recv);
      return( STATUS_SUCCESS );
}

/*
** Calcular el tamano para agregar 3 columnas de tipo DOUBLE.
*/
real_size =  rcvlen + Fneeded(nroFilas*3, sizeof(double));

/*
** Elimina del buffer recibido todo los fields que no interesan.
** si existiesen, no se sabra hasta hacer las pruebas reales con
** el eLink y SAP.
*/
if( Fproj(fbfr_recv, field_arr) == -1 )
{
      userlog("call_RFC_Partidas: Error al llamara a Fproj(fbfr_recv).");
            sprintf(msg, "Error en filtro sobre buffer recibido, Error del tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
            goto _error_rfc;
}

/*
** Se reajusta el tamano del buffer recibido para poner las columnas Debe, Haber y Saldo.
*/
if( (fbfr_recv=(FBFR *)tprealloc((char *)fbfr_recv, real_size) ) == NULL)
{
      userlog("call_RFC_Partidas: Error al llamar a tprealloc(fbfr_recv).");
            sprintf(msg, "Error al ajustar buffer size via tprealloc, Error Tuxedo[%d][%s]" , tperrno, tpstrerror(tperrno) );
            goto _error_rfc;
}
```

```
for(ocurr = 0; ocurr < nroFilas; ocurr++)
{
      _fld_IndicadorDH        = ' ';
      _fld_SapDebeHaber[0]    = ' ';
      _fld_Debe=0, _fld_Haber =0, _fld_Saldo=0;
      _fld_SapSaldo[0]        = '\0';          /* LEN_FLD_SAPSALDO  */
      msg[0]                  = '\0';          /* LEN_MESSAGE       */


      /*  Recupera el Saldo SAP del DebeHaber */
       if ((tux_getField(fbfr_recv, fld_SapDebeHaber, ocurr, _fld_SapDebeHaber, &field_exists) ) ==\
                                                                                                  STATUS_FAIL)
      {
        userlog("call_RFC_Partidas: error en get registro nro[%d] campo fld_SapDebeHaber, Error[%d][%s] ", \
                                                    ocurr, Ferror, Fstrerror(Ferror ));
         sprintf(msg, "fld_SapDebeHaber(%ld), Error get tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
         goto _error_rfc;
      }

      /*  Recupera el indicador de Debe o Haber */
        if ((tux_getField(fbfr_recv, fld_IndicadorDH, ocurr, (char *)&_fld_IndicadorDH, &field_exists) ) \
                                                                                    == STATUS_FAIL)
        {
          userlog("call_RFC_Partidas: error en get registro nro[%d] campo fld_IndicadorDH, Error[%d][%s] ", ocurr,\
                                                                                      Ferror, Fstrerror(Ferror ));
           sprintf(msg, "fld_IndicadorDH(%ld), Error get tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
           goto _error_rfc;
      }

      if(_fld_IndicadorDH == 'S')
            _fld_Debe  = atof(_fld_SapDebeHaber);
      else
            _fld_Haber = atof(_fld_SapDebeHaber);


      /*  Recupera el Monto(Amount) en formato texto */
        if ((tux_getField(fbfr_recv, fld_SapSaldo, ocurr, _fld_SapSaldo, &field_exists) ) == STATUS_FAIL)
        {
          userlog("call_RFC_Partidas: error en get registro nro[%d] campo fld_SapSaldo, Error[%d][%s] ", ocurr,\
                                                    Ferror, Fstrerror(Ferror ));
           sprintf(msg, "fld_SapSaldo(%ld), Error get tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
           goto _error_rfc;
      }
      tux_changeFmtNumber(_fld_SapSaldo);
            _fld_Saldo = atof(_fld_SapSaldo);


      /*
      ** Ahora se ingresan al buffer cada uno de los campos calculados.
      */
      if ((tux_addField(fbfr_recv, fld_Debe , (char *)&_fld_Debe ) == STATUS_FAIL))
      {
            userlog("call_RFC_Partidas: error en add registro nro[%d] campo fld_Debe, Error[%d][%s] ", ocurr, Ferror, \
                                                                                      Fstrerror(Ferror ));
            sprintf(msg, "fld_Debe(%ld), Error add tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
            goto _error_rfc;
      }

      if ((tux_addField(fbfr_recv, fld_Haber , (char *)&_fld_Haber ) == STATUS_FAIL))
      {
            userlog("call_RFC_Partidas: error en add registro nro[%d] campo fld_Haber, Error[%d][%s] ", ocurr,\
                                                            Ferror, Fstrerror(Ferror ));
            sprintf(msg, "fld_Haber(%ld), Error add tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
            goto _error_rfc;
      }


      if ((tux_addField(fbfr_recv, fld_Saldo , (char *)&_fld_Saldo ) == STATUS_FAIL))
      {
             userlog("call_RFC_Partidas: error en add registro nro[%d] campo fld_Saldo, Error[%d][%s] ", \
                                          ocurr, Ferror, Fstrerror(Ferror ));
            sprintf(msg, "fld_Saldo(%ld), Error add tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
            goto _error_rfc;
      }
```

```
	} /* End Ciclo For */

	/*
	** Ahora se procede a sacar la lista de campos que no van al cliente pero
	** que se utilizaron en el for, DEBEHABER Y SALDO.
	*/
	if( Fdelall(fbfr_recv, fld_SapSaldo) == -1)
	{
		userlog("call_RFC_Partidas: error en del campo fld_SapSaldo, Error[%d][%s] ", Ferror, Fstrerror(Ferror ));
		sprintf(msg, "fld_SapSaldo, Error del tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
		goto _error_rfc;
	}
	if( Fdelall(fbfr_recv, fld_SapDebeHaber) == -1)
	{
	  userlog("call_RFC_Partidas: error en del campo fld_SapDebeHaber, Error[%d][%s] ", Ferror, Fstrerror(Ferror ));
	   sprintf(msg, "fld_SapDebeHaber, Error del tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
	   goto _error_rfc;
	}
	*nroFilasOUT = (short)nroFilas;

	/*
	** A estas alturas esta en el buffer fbfr_recv todos los datos que deseamos enviar al cliente
	*/
	*fbfr_auxOUT = fbfr_recv;

	return( STATUS_SUCCESS );

_error_rfc:
	ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , hdrIN );
	tpfree((char *)fbfr_recv);
	return(def_ret);

}



TYPE_STATUS call_RFC_Detalle ( char *CodEmpresaFactIN, char *FolioInternoSapIN, char *FechaContabilizacionIN ,
TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOUT)
{
	long		def_ret	= STATUS_FAIL;

	char		*serviceName = DETALLE_FACTURA;
	char		 msg[LEN_MESSAGE];
	char		 anoEjercicio[5];
	char		_fld_SapImporte[LEN_FLD_SAPIMPORTE];
	char		_fld_SapPosicion[24+1];
	short		_fld_Posicion;
	double		 _fld_Importe;

	long		 fbfr_size   = 1024*4, real_size;
	long		 rcvlen, ret;


	FLDOCC		 nroFilas;
	FLDOCC		 ocurr;

	FBFR		 *fbfr_send;
	FBFR		 *fbfr_recv;
	TYPE_EXISTS	 field_exists;


	static FLDID field_arr[] = { fld_SapPosicion, fld_GlosaDetalle, fld_SapImporte, fld_UsuarioGenerador , BADFLDID };


	*nroFilasOUT	= 0;

	anoEjercicio[0] = '\0';
	ret = tux_getYear(FechaContabilizacionIN, anoEjercicio, FORMAT_YYYYMMDD );


	/*
	** Alocatea espacio e inicializa buffers FML.
	*/
	if( (fbfr_send=(FBFR *)tpalloc(FMLTYPE, NULL, (long)1024)) == NULL)
	{
```

```
        userlog("call_RFC_Detalle: Error en tpalloc [%d][%s]", tperrno, tpstrerror(tperrno));
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , tpstrerror(tperrno) , hdrIN );
        return(def_ret);
}


/*
** Agrega los parametros de entrada para llamar al RFC como un
** servicio mas de Tuxedo.
*/
if ((tux_addField(fbfr_send, fld_CodEmpresaFact , CodEmpresaFactIN ) == STATUS_FAIL))
{
    userlog("call_RFC_Detalle: error en llamada a tux_addField(fld_CodEmpresaFact)  \
                                                                ERROR[%s]",Fstrerror(Ferror));
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
        tpfree((char *)fbfr_send);
        return(def_ret);
}

if ((tux_addField(fbfr_send, fld_FolioInternoSap , FolioInternoSapIN ) == STATUS_FAIL))
{
   userlog("call_RFC_Detalle: error en llamada a tux_addField(fld_FolioInternoSap) ERROR[%s]",Fstrerror(Ferror));
   ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
   tpfree((char *)fbfr_send);
   return(def_ret);
}

if ((tux_addField(fbfr_send, fld_AnoEjercicio , anoEjercicio ) == STATUS_FAIL))
{
        userlog("call_RFC_Detalle: error en llamada a tux_addField(fld_AnoEjercicio) ERROR[%s]",Fstrerror(Ferror));
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , hdrIN );
        tpfree((char *)fbfr_send);
        return(def_ret);
}


/*
** Alocatea e inicializa buffer para recibir los datos
*/
if( (fbfr_recv=(FBFR *)tpalloc(FMLTYPE, NULL, fbfr_size)) == NULL)
{
        userlog("call_RFC_Detalle: Error en tpalloc [%d][%s]", tperrno, tpstrerror(tperrno));
        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , tpstrerror(tperrno) , hdrIN );
        tpfree((char *)fbfr_send);
        return(def_ret);
}
rcvlen = fbfr_size;

if(hdrIN->WithDebug)
{
        printf("\ncall_RFC_Detalle: INICIO: Datos enviados a Elink\n");
        Fprint32(fbfr_send);
        printf("\ncall_RFC_Detalle: FIN:    Datos enviados a Elink\n");
}



/*
** Tpcall al servicio via elink
*/
if ( tpcall(serviceName, (char *)fbfr_send, 0, (char **)&fbfr_recv, &rcvlen, (long)0) == -1)
{
      sprintf(msg, "Error al llamar a SAP via tpcall(%s) , Error del tuxedo[%d][%s]" ,serviceName ,tperrno,\
                                                                                tpstrerror(tperrno) );
       ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , hdrIN );
        userlog("call_RFC_Detalle: Error en tpcall [%d][%s]", tperrno, tpstrerror(tperrno));
        tpfree((char *)fbfr_send);
        tpfree((char *)fbfr_recv);
        return(def_ret);
}
tpfree((char *)fbfr_send);

if(hdrIN->WithDebug)
{
        printf("\ncall_RFC_Detalle: INICIO: Datos recibidos desde Elink\n");
        Fprint32(fbfr_recv);
        printf("\ncall_RFC_Detalle: FIN:    Datos recibidos desde Elink\n");
```

```
}

/*
** Procesamiento de los datos de respuesta desde el RFC de SAP.
*/
nroFilas = Foccur(fbfr_recv, fld_SapPosicion);  /* CJAA 2002JUL12 */

if(nroFilas == 0)
{
      userlog("call_RFC_Detalle: No hay datos asociados a la consulta de Detalle.");
      ret = tux_setMsgHeader( HDR_LEVEL_INFORMATION , "No hay datos asociados a la consulta de Detalle.",\
                                                                                                  hdrIN );

      if( Fproj(fbfr_recv, field_arr) == -1 )
      {
           userlog("call_RFC_Detalle: - Error al llamara a Fproj(fbfr_recv).");
           sprintf(msg, "- Error en filtro sobre buffer recibido, Error del tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
           ret = tux_setMsgHeader( HDR_LEVEL_INFORMATION , msg , hdrIN );
           tpfree((char *)fbfr_recv);
           return(def_ret);
      }

     *fbfr_auxOUT = fbfr_recv;
      return( STATUS_SUCCESS );
}

/*
** Se calcula el tamano para poner el importe que es de tipo DOUBLE
** previa conversion que se debe hacer.
real_size =  rcvlen + Fneeded(nroFilas, sizeof(double)) + Fneeded(nroFilas, 5 );
*/
real_size =  rcvlen + 24 + 12*nroFilas + 8*nroFilas;

/*
** Elimina del buffer recibido todo los fields que no interesan.
** si existiesen, no se sabra hasta hacer las pruebas reales con
** el eLink y SAP.
*/
if( Fproj(fbfr_recv, field_arr) == -1 )
{
      userlog("call_RFC_Detalle: Error al llamara a Fproj(fbfr_recv).");
            sprintf(msg, "Error en filtro sobre buffer recibido, Error del tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
            goto _error_rfc;
}

/*
** Se reajusta el tamano del buffer recibido para poner el Importe convertido de CXhar a Double
*/
if( (fbfr_recv=(FBFR *)tprealloc((char *)fbfr_recv, real_size) ) == NULL)
{
      userlog("call_RFC_Detalle: Error al llamar a tprealloc(fbfr_recv).");
      sprintf(msg, "Error al ajustar buffer size via tprealloc, Error Tuxedo[%d][%s]" , tperrno, tpstrerror(tperrno) );
      goto _error_rfc;
}

for(ocurr = 0; ocurr < nroFilas; ocurr++)
{
      _fld_SapImporte[0]  = '\0';
      _fld_SapPosicion[0] = '\0';

      /*  Recupera el Importe(DMBTR) en formato texto */
        if ((tux_getField(fbfr_recv, fld_SapImporte, ocurr, _fld_SapImporte, &field_exists) ) == STATUS_FAIL)
        {
           userlog("call_RFC_Detalle: error en get registro nro[%d] campo fld_SapImporte, Error[%d][%s] ", ocurr,\
                                                  Ferror, Fstrerror(Ferror ));
           sprintf(msg, "fld_SapImporte(%ld), Error get tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
             goto _error_rfc;
      }
      tux_changeFmtNumber(_fld_SapImporte);
            _fld_Importe = atof(_fld_SapImporte);

      if ((tux_addField(fbfr_recv, fld_Importe , (char *)&_fld_Importe ) == STATUS_FAIL))
      {
           userlog("call_RFC_Detalle: error en add registro nro[%d] campo fld_Importe, Error[%d][%s] ", ocurr,\
                                                        Ferror, Fstrerror(Ferror ));
           sprintf(msg, "fld_Importe(%ld), Error add tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
```

```
                goto _error_rfc;
            }


            /*  Recupera el SapPosicion(DMBTR) en formato texto */
              if ((tux_getField(fbfr_recv, fld_SapPosicion, ocurr, _fld_SapPosicion, &field_exists) ) == STATUS_FAIL)
               {
                    userlog("call_RFC_Detalle: error en get registro nro[%d] campo fld_SapPosicion, Error[%d][%s] ", ocurr,\
                                                                                    Ferror, Fstrerror(Ferror ));
                    sprintf(msg, "fld_SapPosicion(%ld), Error get tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
                   goto _error_rfc;
            }
                    _fld_Posicion = atoi(_fld_SapPosicion);

            if ((tux_addField(fbfr_recv, fld_Posicion , (char *)&_fld_Posicion ) == STATUS_FAIL))
            {
                    userlog("call_RFC_Detalle: error en add registro nro[%d] campo fld_Posicion, Error[%d][%s] ", ocurr,\
                                                                                    Ferror, Fstrerror(Ferror ));
                   sprintf(msg, "fld_Posicion(%ld), Error add tuxedo[%d][%s]" , ocurr , Ferror, Fstrerror(Ferror ) );
                   goto _error_rfc;
            }

      }

      /*
      ** Ahora se procede a sacar la lista de campos que no van al cliente pero
      ** que se utilizaron en el for, fld_SapSaldo
      */
      if( Fdelall(fbfr_recv, fld_SapImporte) == -1)
      {
            userlog("call_RFC_Detalle: error en Fdelall campo fld_SapImporte, Error[%d][%s] ", Ferror, Fstrerror(Ferror ));
                  sprintf(msg, "fld_SapImporte, Error Fdelall tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
                  goto _error_rfc;
      }

      if( Fdelall(fbfr_recv, fld_SapPosicion) == -1)
      {
            userlog("call_RFC_Detalle: error en Fdelall campo fld_SapPosicion, Error[%d][%s] ", Ferror, Fstrerror(Ferror ));
                  sprintf(msg, "fld_SapPosicion, Error Fdelall tuxedo[%d][%s]" , Ferror, Fstrerror(Ferror ) );
                  goto _error_rfc;
      }


      *nroFilasOUT = (short)nroFilas;

      /*
      ** A estas alturas esta en el buffer fbfr_recv todos los datos que deseamos enviar al cliente
      */

      *fbfr_auxOUT = fbfr_recv;

      return( STATUS_SUCCESS );

_error_rfc:
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , hdrIN );
      tpfree((char *)fbfr_recv);
      return(def_ret);

}
```

```
int
#if defined(_STCD_) || defined(____cplusplus)
tpsvrinit(int argc,char **argv)
#else
tpsvrinit(argc,argv)
int argc;
char **argv;
#endif
{
     int auth = 0;
     char strauth[20+1];

     if(ora_connect(argc,argv) == STATUS_FAIL)
          return(-1);

     /*
     ** ACTUALIZA EL NOMBRE DEL SERVIDOR
     */
     strcpy(GV_ServerName,argv[0]);

     if(  (auth=tpchkauth()) == -1 )
          return(-1);

     sprintf(strauth,"SECURITY_LEVEL=%d",auth);
     if(  tuxputenv(strauth) != 0)
          return(-1);

     /*
     ** Todo bien
     */
     return(0);

}

void tpsvrdone()
{

     long ret;

     ret = ora_disconnect();

}
```

ANEXO No 4-8
SERVIDOR DE CAJA

```
/*
**  Archivo: sCaja.pc
**
**  Implementa los servicios Tuxedo  de Caja.
**
**  Nombre del servidor Tuxedo: sCaja
**  Nombre de los Servicios   :
**                          j_DocCtaCteS
**                          j_AnalizaDeudaI
**
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sqlda.h>
#include <sqlcpr.h>
#include <sqlca.h>

#include <atmi.h>
#include <fml32.h>
#include <fml1632.h>
#include <userlog.h>

#include <tux_utils.h>
#include <TblGlobal.fld.h>

/*
** Declaracion de constantes para los largos
** de los campos string MAYOR a 1
*/
#define LEN_FLD_RUTCLIENTE         10 + 1
#define LEN_FLD_NROSERVICIO        16 + 1
#define LEN_FLD_TIPODOCCAJA         3 + 1
#define LEN_FLD_FOLIODOCUMENTO     18 + 1
#define LEN_FLD_NROCLIENTE          9 + 1
#define LEN_FLD_NROCUENTA           9 + 1
#define LEN_FLD_SAPID              10 + 1

#define LEN_FLD_TIPODOCCAJA           3 + 1
#define LEN_FLD_GLOSATIPODOC       20 + 1
#define LEN_FLD_CODEMPRESAFACT    4 + 1
#define LEN_FLD_FOLIODOCUMENTO   18 + 1
#define LEN_FLD_CODBANCO             3 + 1
#define LEN_FLD_ESTADOCTECTA       15 + 1
#define LEN_FLD_FECHAVCTOPAGO      8 + 1
#define LEN_FLD_FOLIOINTERNOSAP    10 + 1
#define LEN_FLD_SAPID                10 + 1
#define LEN_FLD_NOMBRECLIENTE      40 + 1
#define LEN_FLD_NUMERORUTCLIENTE   10 + 1

#define LEN_FLD_SUCURSAL          10 + 1
#define LEN_FLD_CAJA               3 + 1
#define LEN_FLD_HOST               8 + 1
#define LEN_FLD_FECHARECAUDACION    8 + 1
#define LEN_FLD_OBSERVACION        40 + 1

#define LEN_FLD_SAPSALDO          10 + 1
#define LEN_FLD_SAPVALORCCO       10 + 1
#define LEN_FLD_SAPVALORIMPORTE    10 + 1

/*
** INserta reposicion
*/
#define LEN_FLD_TIPOTRANSACCION    10 + 1
#define LEN_FLD_ESTADOTRAN         15 + 1


/*
** Define los estados asquerosos que maneja Bellsouth en CAJA
** parece increible.
*/
#define ESTADO_BAD_CREDIT          "Bad Credit"
#define ESTADO_A_BAD_CREDIT        "A Bad Credit"
#define ESTADO_NO_PAGO             "No Pago"
#define ESTADO_A_NO_PAGO           "A No Pago"
```

```
#define TIPODOCCAJA_BOLETA        "B"
#define TIPODOCCAJA_FACTURA       "F"
#define TIPODOCCAJA_OTRO          " "

#define RFC_DOCCTACTE             "Tx_jDocCtaCte"
#define MAX_DOCS_POR_CLICTA       500
#define MAX_BUFFER_CO             2000

/*
** Lista enlazada para mantener los grupos distintos
*/
struct gruposClienteCuenta
{
    char   NroCliente[LEN_FLD_NROCLIENTE];            /* Nro Cliente  */
    char   NroCuenta[LEN_FLD_NROCUENTA];              /* Nro Cuenta   */
    char   EstadoCteCta[LEN_FLD_ESTADOCTECTA]; /* Estado del grupo del grupo Cliente-Cuenta.    */
    char   TipoDocCaja[LEN_FLD_TIPODOCCAJA];     /* Tipo Documento Caja.                          */
    char   SapId[LEN_FLD_SAPID];                 /* Sap ID del Cliente Cuenta                     */
    double TotalPagos;                           /* Suma de de Pagos del Cliente Cuenta           */
    char   FechaRecaudacion[LEN_FLD_FECHARECAUDACION];
                                          /* Fecha de recaudacion de  la primera Boleta o Factura  */
    char   CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];    /* Codigo Empresa Facturacion            */

    char   BoF[LEN_FLD_TIPODOCCAJA];              /* Char que indica que Doc Es [B,F,' ']          */
    short  ExisteBoF;                             /* 1=Existe Boleta o Factura  0, NO existe       */
    FLDOCC OcurrBoF;                              /* Ocurrencia de donde se saco la Primera Boleta o Factura  */

    FLDOCC Ocurrencias[MAX_DOCS_POR_CLICTA];   /* Lista de ocurrencias que forman parte del grupo Cli-Cta. */
    short  TotalOcurrencias;
    struct gruposClienteCuenta *next;
};
typedef struct gruposClienteCuenta GRUPOS_CLIENTECUENTA;



typedef char    type_NroCliente[LEN_FLD_NROCLIENTE];
typedef char    type_NroCuenta[LEN_FLD_NROCUENTA];
typedef char    type_CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];

EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR   loc_MessageSP[LEN_MESSAGE];

    char          GV_ServerName[78+1];

    /*
    ** Equivalencia de tipos
    */
    EXEC SQL TYPE type_NroCliente     IS STRING(LEN_FLD_NROCLIENTE)     REFERENCE;
    EXEC SQL TYPE type_NroCuenta      IS STRING(LEN_FLD_NROCUENTA)      REFERENCE;
    EXEC SQL TYPE type_CodEmpresaFact  IS STRING(LEN_FLD_CODEMPRESAFACT)  REFERENCE;

    type_NroCliente      ora_fld_NroCliente[MAX_BUFFER_CO];
    type_NroCuenta       ora_fld_NroCuenta[MAX_BUFFER_CO];
    type_CodEmpresaFact   ora_fld_CodEmpresaFact[MAX_BUFFER_CO];
    long                  o_fld_NroFilasCO;

    /*
    ** Campos utilizados en el servicio j_DocCtaCteS
    */
    char     i_fld_TipoConsultaCj;
    char     i_fld_CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];
    char     i_fld_RutCliente[LEN_FLD_RUTCLIENTE];
    char     i_fld_NroServicio[LEN_FLD_NROSERVICIO];
    char     i_fld_TipoDocCaja[LEN_FLD_TIPODOCCAJA];
    char     i_fld_FolioDocumento[LEN_FLD_FOLIODOCUMENTO];
    char     i_fld_NroCliente[LEN_FLD_NROCLIENTE];
    char     i_fld_NroCuenta[LEN_FLD_NROCUENTA];
    char     i_fld_SapId[LEN_FLD_SAPID];

    char     co_fld_TipoDocCaja[LEN_FLD_TIPODOCCAJA];
    char     co_fld_GlosaTipoDoc[LEN_FLD_GLOSATIPODOC];
    char     co_fld_CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];
    char     co_fld_FolioDocumento[LEN_FLD_FOLIODOCUMENTO];
    char     co_fld_CodBanco[LEN_FLD_CODBANCO];
    char     co_fld_NroCliente[LEN_FLD_NROCLIENTE];
    char     co_fld_NroCuenta[LEN_FLD_NROCUENTA];
    char     co_fld_NroServicio[LEN_FLD_NROSERVICIO];
```

```
        double    co_fld_Saldo;
        char      co_fld_EstadoCteCta[LEN_FLD_ESTADOCTECTA];
        char      co_fld_FechaVctoPago[LEN_FLD_FECHAVCTOPAGO];
        char      co_fld_FolioInternoSap[LEN_FLD_FOLIOINTERNOSAP];
        char      co_fld_SapId[LEN_FLD_SAPID];
        double    co_fld_ValorCCO;
        double    co_fld_ValorImporte;
        char      co_fld_NombreCliente[LEN_FLD_NOMBRECLIENTE];

        /*
        ** Campos utilizados en el servicio j_AnalizaDeudaI
        */
        char      ci_fld_NroCliente[LEN_FLD_NROCLIENTE];
        char      ci_fld_NroCuenta[LEN_FLD_NROCUENTA];
        char      ci_fld_SapId[LEN_FLD_SAPID];
        char      ci_fld_IndCtaControlada;
        char      ci_fld_NroServicio[LEN_FLD_NROSERVICIO];
        char      ci_fld_CodEmpresaFact[LEN_FLD_CODEMPRESAFACT];
        char      ci_fld_Sucursal[LEN_FLD_SUCURSAL];
        char      ci_fld_Caja[LEN_FLD_CAJA];
        char      ci_fld_Host[LEN_FLD_HOST];
        char      ci_fld_FechaRecaudacion[LEN_FLD_FECHARECAUDACION];
        float     ci_fld_TransDia;
        float     ci_fld_TransHist;
        char      ci_fld_TipoDocCaja[LEN_FLD_TIPODOCCAJA];
        char      ci_fld_FolioDocumento[LEN_FLD_FOLIODOCUMENTO];
        double    ci_fld_MontoRecaudado;
        char      ci_fld_Observacion[LEN_FLD_OBSERVACION];

EXEC SQL END DECLARE SECTION;




/*
** Prototipo de funciones de llamadas a RFC
*/
TYPE_STATUS  call_RFC_DocCtaCte      ( short consPorNroServicioIN, char *NroServicioIN, char TipoConsultaCjIN,
char *CodEmpresaFactIN, char *RutClienteIN, char *TipoDocCajaIN , char *FolioDocumentoIN, char *NroClienteIN, char
*NroCuentaIN,char *SapIdIN ,TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOUT);

TYPE_STATUS  call_RFC_DocCtaCteV1    ( char TipoConsultaCjIN, char *CodEmpresaFactIN, char *RutClienteIN,
char *TipoDocCajaIN , char *FolioDocumentoIN, char *NroClienteIN, char *NroCuentaIN,char *SapIdIN
,TYPE_HEADERMESSAGE *hdrIN ,FBFR **fbfr_auxOUT , short *nroFilasOUT);

TYPE_STATUS  call_crearGrupos        ( GRUPOS_CLIENTECUENTA **grp  , FBFR *fbfrIN ,
TYPE_HEADERMESSAGE *hdrIN , short *nroGruposOUT );

TYPE_STATUS  call_Reposicion         ( GRUPOS_CLIENTECUENTA *grpIN , FBFR *fbfrIN ,
TYPE_HEADERMESSAGE *hdrIN );

TYPE_STATUS  call_InsertReposicion   ( GRUPOS_CLIENTECUENTA *grpIN , FBFR *fbfrIN ,
TYPE_HEADERMESSAGE *hdrIN);

TYPE_STATUS  call_Recarga            ( GRUPOS_CLIENTECUENTA *grpIN , FBFR *fbfrIN , TYPE_HEADERMESSAGE
*hdrIN , FLDOCC ocurrIN , char *TipoDocCajaIN);

TYPE_STATUS  call_InsertRecarga      ( GRUPOS_CLIENTECUENTA *grpIN , FBFR *fbfrIN ,
TYPE_HEADERMESSAGE *hdrIN , FLDOCC ocurrIN , char *TipoDocCajaIn);




#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
j_DocCtaCteS(TPSVCINFO *rqst)
#else
j_DocCtaCteS(rqst)
TPSVCINFO *rqst;
#endif
{
      TYPE_STATUS   ora_status = ORA_STATUS_SUCCESS;
      char          ora_message[LEN_MESSAGE];
      long          SQLCODE=0;
      short         idx;
```

```
	short		nro_filas=0;
	short		nro_filasAcumuladas=0;
	char		msg[LEN_MESSAGE];
	TYPE_STATUS	ret;
	TYPE_EXISTS	field_exists;
	long		totalClientes = 0;
	short		consPorNroServicio;
	short		prevAlloc     = 0;  /* INdica si se ha alocateado previamente con tpalloc antes deun tprealloc*/

	FBFR		*fbfr     = (FBFR *)0;
	FBFR		*fbfr_aux = (FBFR *)0;
	FBFR		*fbfr_tmp = (FBFR *)0;
	FLDLEN		len;
	long		size_acumulado = 0;
	long		size_tmp       = 0;
	FLDOCC		ocurr=0;

	TYPE_HEADERMESSAGE hdr;

	char		priv_fld_StatusUser;


	consPorNroServicio = 0;

	/*******************************************************
	  PASE LO QUE PASE SIGA, NO SALIR CON EXIT O ALGO ASI
	*******************************************************/
	EXEC SQL WHENEVER SQLERROR  CONTINUE;
	EXEC SQL WHENEVER NOT FOUND CONTINUE;


	/*******************************************************
	     OBTIENE BUFFER FML32 DE LOS DATOS DE ENTRADA
	*******************************************************/
	fbfr = (FBFR *)rqst->data;


	/*******************************************************
	   INICIALIZA EL HEADER DEL MENSAJE TUXEDO
	   CON VALORES POR DEFECTO
	*******************************************************/
	ret = tux_initHeader(&hdr, rqst->name );

	/*******************************************************
	   CHEQUEO DE PRIVILEGIOS

	*******************************************************/
#if defined(WITH_PRIVI)
	priv_fld_StatusUser = ' ';
	ret = tux_CheckPrivi(GV_ServerName, rqst, &hdr, &priv_fld_StatusUser );
	if(ret == STATUS_FAIL)
	{
	  userlog("[%s]: Usuario sin acceso al Servicio[%s] Causa[%s]Status[%c]" , rqst->name , rqst->name,\
                                                                hdr.o_fld_hdr_MsgText,priv_fld_StatusUser);
	    sprintf(msg,"Usuario sin acceso al Servicio[%s] Causa[%s]"   , rqst->name, hdr.o_fld_hdr_MsgText);
	    ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
	    goto _error;
	}
#endif

	/*******************************************************
	   OBTIENE EL HEADER DEL MENSAJE TUXEDO
	*******************************************************/
	if (tux_getHeader(rqst, &hdr ) == STATUS_FAIL )
	{
	    userlog("[%s]: Fallo llamada a tux_getHeader" , rqst->name );
	    ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Error al obtener Header del mensaje de entrada" , &hdr );
	    goto _error;
	}

	/*******************************************************
	   OBTIENE LOS PARAMETROS DE ENTRADA
	   PROPIOS DEL SERVICIO.
	*******************************************************/
	 ocurr = 0;
	 if (( tux_getField(fbfr, fld_TipoConsultaCj, ocurr, (char *)&i_fld_TipoConsultaCj, &field_exists) ) == STATUS_FAIL)
	 {
	      if(field_exists == FIELD_NOT_EXISTS)
```

```
        {
            userlog("[%s]:ADVERTENCIA: campo fld_TipoConsultaCj  no fue recibido en msg. de entrada", \
                                                                                    rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_TipoConsultaCj no fue recibido \
                                                                                en msg. de entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_TipoConsultaCj)=[%d][%s]", rqst->name ,\
                                                                    Ferror, Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_TipoConsultaCj [%c]", rqst->name , i_fld_TipoConsultaCj );


    i_fld_CodEmpresaFact[0] = '\0';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_CodEmpresaFact, ocurr, i_fld_CodEmpresaFact, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_CodEmpresaFact no fue recibido en msg. de entrada",\
                                                                                    rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_CodEmpresaFact no fue recibido en msg.\
                                                                                de entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_CodEmpresaFact)=[%d][%s]",\
                                                                rqst->name , Ferror, Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_CodEmpresaFact [%s]", rqst->name , i_fld_CodEmpresaFact );


    i_fld_RutCliente[0] = '\0';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_RutCliente, ocurr, i_fld_RutCliente, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_RutCliente  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_RutCliente no fue recibido en msg. De\
                                                                                    entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_RutCliente)=[%d][%s]", rqst->name , Ferror,\
                                                                                Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_RutCliente [%s]", rqst->name , i_fld_RutCliente );
    if( strlen( i_fld_RutCliente ) > 2 && strcmp(i_fld_RutCliente,"0") != 0 && strcmp(i_fld_RutCliente,"0000000000") != 0
) /* Supongo que 1-9 es el minimo RUT posible, largo 2 sin guion y con CEROS */
    {
        while(i_fld_RutCliente[0] == '0')
        {
            len = strlen(i_fld_RutCliente);
            for( idx=0; idx < (len-1) ; idx++ )
            {
                i_fld_RutCliente[idx] = i_fld_RutCliente[idx+1];
            }
            i_fld_RutCliente[idx] = '\0';
        }
    }
```

```
	i_fld_NroServicio[0] = '\0';
	ocurr = 0;
	if (( tux_getField(fbfr, fld_NroServicio, ocurr, i_fld_NroServicio, &field_exists) ) == STATUS_FAIL)
	{
		if(field_exists == FIELD_NOT_EXISTS)
		{
			userlog("[%s]:ADVERTENCIA: campo fld_NroServicio  no fue recibido en msg. de entrada", rqst->name );
			ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_NroServicio no fue recibido en msg. De\
entrada" , &hdr );
		}
		else
		{
			userlog("[%s]: ERROR en llamada a tux_getField(fld_NroServicio)=[%d][%s]", rqst->name , Ferror,\
Fstrerror(Ferror) );
			ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
		}
		goto _error;
	}
	else if(hdr.WithDebug)
		userlog("[%s]:Se recibio fld_NroServicio [%s]", rqst->name , i_fld_NroServicio );


	i_fld_TipoDocCaja[0] = '\0';
	ocurr = 0;
	if (( tux_getField(fbfr, fld_TipoDocCaja, ocurr, i_fld_TipoDocCaja, &field_exists) ) == STATUS_FAIL)
	{
		if(field_exists == FIELD_NOT_EXISTS)
		{
			userlog("[%s]:ADVERTENCIA: campo fld_TipoDocCaja  no fue recibido en msg. de entrada", rqst->name
);
			ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_TipoDocCaja no fue recibido en msg. De\
entrada" , &hdr );
		}
		else
		{
			userlog("[%s]: ERROR en llamada a tux_getField(fld_TipoDocCaja)=[%d][%s]", rqst->name , Ferror,\
Fstrerror(Ferror) );
			ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
		}
		goto _error;
	}
	else if(hdr.WithDebug)
		userlog("[%s]:Se recibio fld_TipoDocCaja [%s]", rqst->name , i_fld_TipoDocCaja );


	i_fld_FolioDocumento[0] = '\0';
	ocurr = 0;
	if (( tux_getField(fbfr, fld_FolioDocumento, ocurr, i_fld_FolioDocumento, &field_exists) ) == STATUS_FAIL)
	{
		if(field_exists == FIELD_NOT_EXISTS)
		{
			userlog("[%s]:ADVERTENCIA: campo fld_FolioDocumento  no fue recibido en msg. de entrada", \
rqst->name );
			ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_FolioDocumento no fue recibido en \
msg. de entrada" , &hdr );
		}
		else
		{
			userlog("[%s]: ERROR en llamada a tux_getField(fld_FolioDocumento)=[%d][%s]", rqst->name , Ferror,\
Fstrerror(Ferror) );
			ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
		}
		goto _error;
	}
	else if(hdr.WithDebug)
		userlog("[%s]:Se recibio fld_FolioDocumento [%s]", rqst->name , i_fld_FolioDocumento );


	i_fld_NroCliente[0] = '\0';
	ocurr = 0;
	if (( tux_getField(fbfr, fld_NroCliente, ocurr, i_fld_NroCliente, &field_exists) ) == STATUS_FAIL)
	{
		if(field_exists == FIELD_NOT_EXISTS)
		{
```

```
            userlog("[%s]:ADVERTENCIA: campo fld_NroCliente  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_NroCliente no fue recibido en msg. De\
                                                                                                    entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_NroCliente)=[%d][%s]", rqst->name , Ferror,\
                                                                                                    Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_NroCliente [%s]", rqst->name , i_fld_NroCliente );


    i_fld_NroCuenta[0] = '\0';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_NroCuenta, ocurr, i_fld_NroCuenta, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_NroCuenta  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_NroCuenta no fue recibido en msg. De\
                                                                                                    entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_NroCuenta)=[%d][%s]", rqst->name , Ferror,\
                                                                                                    Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_NroCuenta [%s]", rqst->name , i_fld_NroCuenta );


    i_fld_SapId[0] = '\0';
    ocurr = 0;
    if (( tux_getField(fbfr, fld_SapId, ocurr, i_fld_SapId, &field_exists) ) == STATUS_FAIL)
    {
        if(field_exists == FIELD_NOT_EXISTS)
        {
            userlog("[%s]:ADVERTENCIA: campo fld_SapId  no fue recibido en msg. de entrada", rqst->name );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_SapId no fue recibido en msg. de \
                                                                                        entrada" , &hdr );
        }
        else
        {
            userlog("[%s]: ERROR en llamada a tux_getField(fld_SapId)=[%d][%s]", rqst->name , Ferror,\
                                                                                        Fstrerror(Ferror) );
            ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror(Ferror) , &hdr );
        }
        goto _error;
    }
    else if(hdr.WithDebug)
        userlog("[%s]:Se recibio fld_SapId [%s]", rqst->name , i_fld_SapId );

    /*
    ** Si TipoConsultaCj = 1 , entonces fld_FolioDocumento debe valido, no blancos, no cero,
    */
    if ( i_fld_TipoConsultaCj == '1')
    {
        if( atol(i_fld_FolioDocumento) == 0  )
        {
            sprintf(msg,"fld_FolioDocumento invalido [%s]", i_fld_FolioDocumento);
            userlog("[%s]: fld_FolioDocumento[%s] invalido.", rqst->name , i_fld_FolioDocumento );
            ret = tux_setMsgHeader( HDR_LEVEL_WARNING , msg , &hdr );
            goto _error;
        }
        if( (strcmp(i_fld_TipoDocCaja," ") == 0) || (strlen(i_fld_TipoDocCaja) == 0) )
```

```
	{
		sprintf(msg,"fld_TipoDocCaja invalido [%s]", i_fld_TipoDocCaja);
		userlog("[%s]: fld_TipoDocCaja[%s] invalido.", rqst->name , i_fld_TipoDocCaja );
		ret = tux_setMsgHeader( HDR_LEVEL_WARNING , msg , &hdr );
		goto _error;
	}
	if( (strcmp(i_fld_CodEmpresaFact," ") == 0) || (strlen(i_fld_CodEmpresaFact) == 0) )
	{
		sprintf(msg,"fld_CodEmpresaFact invalido [%s]", i_fld_CodEmpresaFact);
		userlog("[%s]: fld_CodEmpresaFact[%s] invalido.", rqst->name , i_fld_CodEmpresaFact );
		ret = tux_setMsgHeader( HDR_LEVEL_WARNING , msg , &hdr );
		goto _error;
	}
}

/*
** Valida que exista un numero de servicio valido.
*/
totalClientes = 0;
if(  atof(i_fld_NroServicio) > 0 )
{
		strcpy(i_fld_RutCliente, " ");
	i_fld_NroCliente[0] = '\0';
	i_fld_NroCuenta[0]  = '\0';

	consPorNroServicio = 1;


	memset((char *)loc_MessageSP.arr,0, LEN_MESSAGE);
	loc_MessageSP.len  = 0;
	sprintf( (char *)loc_MessageSP.arr,"%04d", (hdr.i_fld_hdr_MaxFilasCO + 1) );
	loc_MessageSP.len  = strlen((char *)loc_MessageSP.arr);
	o_fld_NroFilasCO   = 0;
	EXEC SQL EXECUTE
		BEGIN j_ListaCliCtaS_Pkg.j_ListaCliCtaS
			( :loc_MessageSP
			 ,:i_fld_NroServicio
			 ,:o_fld_NroFilasCO
			 ,:ora_fld_NroCliente
			 ,:ora_fld_NroCuenta
			 ,:ora_fld_CodEmpresaFact
			);
		END;
	END-EXEC;


	ret = ora_checkStatusSql((char *)loc_MessageSP.arr , &ora_status , ora_message);
	if(ora_status != ORA_STATUS_SUCCESS)
	{
		userlog("[%s] Error al llamar a j_ListaCliCtaS_Pkg.j_ListaCliCtaS con NroServicio[%s]", rqst->name,\
                                                                                                i_fld_NroServicio);
		ret = tux_setMsgHeader( ora_status , ora_message , &hdr );
		goto  _error;
	}

	totalClientes = o_fld_NroFilasCO;

}
else if(  atof(i_fld_SapId) > 0 )
{
	if( strlen(i_fld_CodEmpresaFact) == 0 || strcmp(i_fld_CodEmpresaFact, " ") == 0  ||
                                                        strcmp(i_fld_CodEmpresaFact, "    ")== 0 )
	{
		sprintf(msg,"%s: fld_CodEmpresaFact invalido [%s]", rqst->name , i_fld_CodEmpresaFact);
		userlog("[%s]: fld_CodEmpresaFact[%s] invalido.", rqst->name , i_fld_CodEmpresaFact );
		ret = tux_setMsgHeader( HDR_LEVEL_WARNING , msg , &hdr );
		goto _error;
	}
	strcpy(i_fld_RutCliente, " ");
}



/*
** Llama a la RFC
*/
if( consPorNroServicio == 0 )  /* Se consulto por cualquier cosa menos por numero de servicio */
{
			if(  call_RFC_DocCtaCte( consPorNroServicio,i_fld_NroServicio , i_fld_TipoConsultaCj,
			i_fld_CodEmpresaFact , i_fld_RutCliente, i_fld_TipoDocCaja , i_fld_FolioDocumento,
```

```
                i_fld_NroCliente, i_fld_NroCuenta, i_fld_SapId , &hdr ,(FBFR **)&fbfr_aux , &nro_filas) ==
                STATUS_FAIL )
        {
            userlog("[%s] A: MSG_ERROR hdr: [%ld][%s]", rqst->name , hdr.o_fld_hdr_MsgCode,\
                                                    hdr.o_fld_hdr_MsgText );
            userlog("[%s] A: Error al llamr a RFC de SAP via call_RFC_DocCtaCte. consPorNroServicio[%d]",\
                                                        rqst->name, consPorNroServicio );
            goto  _error;
        }
}
else
{
        /*
        ** Se consulto por Numero de Servicio
        ** y puede existir mas de un par cliente cuenta.
        */
        size_tmp       = 0;
        size_acumulado = 0;
        prevAlloc      = 0;
        nro_filasAcumuladas=0;
    for(idx=0; idx < totalClientes; idx++ )
    {

        strcpy(i_fld_NroCliente     , ora_fld_NroCliente[idx]     );
        strcpy(i_fld_NroCuenta      , ora_fld_NroCuenta[idx]      );
        strcpy(i_fld_CodEmpresaFact , ora_fld_CodEmpresaFact[idx] );

        nro_filas = 0;


          if(  call_RFC_DocCtaCte( consPorNroServicio,i_fld_NroServicio , i_fld_TipoConsultaCj,
          i_fld_CodEmpresaFact , i_fld_RutCliente, i_fld_TipoDocCaja , i_fld_FolioDocumento,
          i_fld_NroCliente, i_fld_NroCuenta, i_fld_SapId , &hdr ,(FBFR **)&fbfr_tmp , &nro_filas) ==
          STATUS_FAIL )
        {
            if(hdr.o_fld_hdr_MsgCode != HDR_LEVEL_INFORMATION )
            {
                userlog("[%s] A1: MSG_ERROR hdr: [%ld][%s]", rqst->name , hdr.o_fld_hdr_MsgCode,\
                                                                hdr.o_fld_hdr_MsgText );
                userlog("[%s] A2: Error al llamr a RFC de SAP via call_RFC_DocCtaCte. idx[%d]", \
                                                                    rqst->name , idx );
                goto  _error;
            }
            else
                userlog("[%s] A3: MSG_ERROR hdr: [%ld][%s]", rqst->name , hdr.o_fld_hdr_MsgCode,\
                                                                hdr.o_fld_hdr_MsgText );
        }

        /*
        ** A medida que llegan filas se copian al buffer fbfr_aux para acumular
        */
        if( nro_filas > 0 )
        {
            nro_filasAcumuladas = nro_filasAcumuladas + nro_filas;
                /* Calcula el tamano del buffer recibido */
                if(  (size_tmp = Fsizeof32( fbfr_tmp )) == -1 )
                {
                userlog("[%s] Error al llamar a Fsizeof32(fbfr_tmp) idx[%d] Tuxedo[%d][%s]", rqst->name ,\
                                                        idx, Ferror, Fstrerror(Ferror) );
                sprintf(msg,"[%s] Error al llamar a Fsizeof32(fbfr_tmp) idx[%d] Tuxedo[%d][%s]", \
                                                    rqst->name , idx, Ferror, Fstrerror(Ferror) );
                ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
                if(fbfr_tmp) tpfree((char *)fbfr_tmp);
                if(fbfr_aux) tpfree((char *)fbfr_aux);
                goto  _error;
                }

                size_acumulado = size_acumulado + size_tmp;

                if(prevAlloc > 0)
                {

                    if( (fbfr_aux=(FBFR32 *)tprealloc((char *)fbfr_aux, size_acumulado )) == NULL )
                    {
                    userlog("[%s] Error al llamar a tprealloc(fbfr_aux) idx[%d] Tuxedo[%d][%s]",\
                                                        rqst->name , idx, tperrno, tpstrerror(tperrno) );
```

```
                                        sprintf(msg,"[%s] Error al llamar a tprealloc(fbfr_aux) idx[%d] Tuxedo[%d][%s]", \
                                                                rqst->name , idx, tperrno, tpstrerror(tperrno) );
                                        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
                                        if(fbfr_tmp) tpfree((char *)fbfr_tmp);
                                        if(fbfr_aux) tpfree((char *)fbfr_aux);
                                        goto  _error;
                                         }
                                }
                                else
                                {

                                         if( (fbfr_aux=(FBFR32 *)tpalloc(FMLTYPE, NULL ,size_acumulado )) == NULL )
                                         {
                                        userlog("[%s] Error al llamar a tpalloc(fbfr_aux) idx[%d] Tuxedo[%d][%s]", \
                                                                rqst->name , idx, tperrno, tpstrerror(tperrno) );
                                        sprintf(msg,"[%s] Error al llamar a tpalloc(fbfr_aux) idx[%d] Tuxedo[%d][%s]", \
                                                                rqst->name , idx, tperrno, tpstrerror(tperrno) );
                                        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
                                        if(fbfr_tmp) tpfree((char *)fbfr_tmp);
                                        if(fbfr_aux) tpfree((char *)fbfr_aux);
                                        goto  _error;
                                         }
                                         prevAlloc = prevAlloc + 1;
                                }

                                 if( Fconcat32(fbfr_aux, fbfr_tmp) == -1)
                                 {
                                userlog("[%s] Error al llamar a Fconcat32(fbfr_aux, fbfr_tmp) idx[%d] Tuxedo[%d][%s]", \
                                                                rqst->name , idx, Ferror, Fstrerror(Ferror) );
                                sprintf(msg,"[%s] Error al llamar a Fconcat32(fbfr_aux, fbfr_tmp) idx[%d] Tuxedo[%d][%s]",\
                                                                 rqst->name , idx, Ferror, Fstrerror(Ferror) );
                                ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
                                if(fbfr_tmp) tpfree((char *)fbfr_tmp);
                                if(fbfr_aux) tpfree((char *)fbfr_aux);
                                goto _error;
                                 }

                                 /* Cada vez se va liberando el buffer temporal.*/
                                 tpfree((char *)fbfr_tmp);
                                 fbfr_tmp = (FBFR32 *)0;
                        }


                }
        } /* Fin del if() de la consulta por Numero de Servicio */

        /*
        ** Actualiza el numero de filas retornadas por el servicio.
        */
        if(consPorNroServicio == 0)
                hdr.o_fld_hdr_NroFilasCO = nro_filas;
        else
        {       /* Se consulto por Numero de Servicio y el total de filas es el acumulado
                ** para los distintos pares de Cliente/Cuenta/EmpresaFacturadora.
                */
                hdr.o_fld_hdr_NroFilasCO = nro_filasAcumuladas;
                        if(nro_filasAcumuladas > 0)
                        {
                                hdr.o_fld_hdr_MsgCode = 0;
                                strcpy(hdr.o_fld_hdr_MsgText, " ");
                        }
        }

_exit:
        ret = tux_setData(rqst, fbfr_aux,  hdr);
        tpfree((char *)fbfr_aux);
        if (ret == STATUS_FAIL)
                tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
        else
                tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);

_error:
        ret = tux_setDataError(rqst, hdr);
        if (ret == STATUS_FAIL)
                tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
        else
                tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);
```

```
}


/*
*****************************************************************************************************************
**
** Servicio j_AnalizaDeudaI
**
*****************************************************************************************************************
*/
#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
j_AnalizaDeudaI(TPSVCINFO *rqst)
#else
j_AnalizaDeudaI(rqst)
TPSVCINFO *rqst;
#endif
{
	TYPE_STATUS   ora_status;
	char          ora_message[LEN_MESSAGE];
	short         idx;
	short         nro_filas=0,nro_grupos=0;
	TYPE_STATUS   ret;
	TYPE_EXISTS   field_exists;

	FBFR32       *fbfr=(FBFR32 *)0, *fbfr_aux=(FBFR32 *)0;
	FLDLEN32      len;
	FLDOCC32      ocurr;

	char          priv_fld_StatusUser;
	char          msg[LEN_MESSAGE];

	TYPE_HEADERMESSAGE    hdr;
	GRUPOS_CLIENTECUENTA *grupos = (GRUPOS_CLIENTECUENTA *)0;
	GRUPOS_CLIENTECUENTA *grp    = (GRUPOS_CLIENTECUENTA *)0;

	/*******************************************************
	  PASE LO QUE PASE SIGA, NO SALIR CON EXIT O ALGO ASI
	*******************************************************/
	EXEC SQL WHENEVER SQLERROR  CONTINUE;
	EXEC SQL WHENEVER NOT FOUND CONTINUE;


	/*******************************************************
	     OBTIENE BUFFER FML32 DE LOS DATOS DE ENTRADA
	*******************************************************/
	fbfr = (FBFR32 *)rqst->data;


	/*******************************************************
	   INICIALIZA EL HEADER DEL MENSAJE TUXEDO
	   CON VALORES POR DEFECTO
	*******************************************************/
	ret = tux_initHeader(&hdr, rqst->name );

	/*******************************************************
	   CHEQUEO DE PRIVILEGIOS

	*******************************************************/
#if defined(WITH_PRIVI)
	priv_fld_StatusUser = ' ';
	ret = tux_CheckPrivi(GV_ServerName, rqst, &hdr, &priv_fld_StatusUser );
	if(ret == STATUS_FAIL)
	{
		userlog("[%s]: Usuario sin acceso al Servicio[%s] Causa[%s]Status[%c]" , rqst->name , rqst->name,\
                                                                hdr.o_fld_hdr_MsgText,priv_fld_StatusUser);
		sprintf(msg,"Usuario sin acceso al Servicio[%s] Causa[%s]"   , rqst->name, hdr.o_fld_hdr_MsgText);
		ret = tux_setMsgHeader( HDR_LEVEL_ERROR , msg , &hdr );
		goto _error;
	}
#endif

	/*******************************************************
	   OBTIENE EL HEADER DEL MENSAJE TUXEDO
	*******************************************************/
	if (tux_getHeader(rqst, &hdr ) == STATUS_FAIL )
```

```
{
      userlog("[%s]: Fallo llamada a tux_getHeader" , rqst->name );
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Error al obtener Header del mensaje de entrada" , &hdr );
      goto _error;
}


/*****************************************************************************************
   OBTIENE LOS DISTINTOS GRUPOS DE CLIENTES CUENTA Y LOS DEVUELVE EN LA LISTA grupos

*****************************************************************************************/
if( call_crearGrupos(  (GRUPOS_CLIENTECUENTA **)&grupos ,  fbfr , &hdr , &nro_grupos ) == STATUS_FAIL )
{
      userlog("[%s]: Fallo llamada a crearGrupos" , rqst->name );
      ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Error al crear lista de grupos Cliente - Cuenta" , &hdr );
      goto _error;
}

/*********************************************************************
   A CONTINUACION SE RECORRE LA LISTA  ENLAZADA CON LOS
   DISTINTOS GRUPOS CLIENTE-CUENTA ENCONTRADOS PARA EFECTUAR LA

   REPOSICION

*********************************************************************/
if(hdr.WithDebug)
      userlog("[%s] ***** INICIO PROCESO DE REPOSICION ***** ", rqst->name );
if( (nro_grupos > 0) && grupos )
{


      grp = grupos;
      while( grp )
      {

            if(hdr.WithDebug)
                  userlog("[%s]: Recorriendo NODOS en proceso de REPOSICION en EstadoCteCta[%s]",\
                                                                        rqst->name , grp->EstadoCteCta );
            if(     (strcmp(grp->EstadoCteCta , ESTADO_BAD_CREDIT   ) == 0)
                || (strcmp(grp->EstadoCteCta , ESTADO_NO_PAGO      ) == 0)

              )
            {

                  if(hdr.WithDebug)
                        userlog("[%s]:            Se llamara a  call_Reposicion [%s]", rqst->name , grp->EstadoCteCta );

                  if( call_Reposicion( grp , fbfr , &hdr ) == STATUS_FAIL )
                  {
                        userlog("[%s]: Fallo llamada a call_Reposicion" , rqst->name );
                        ret = tux_setMsgHeader( HDR_LEVEL_ERROR , "Fallo llamada a call_Reposicion" , &hdr );
                        goto _error;
                  }

            } /* Fin del if que evalua los 4 estados necesarios para el proceso  */

            grp = grp->next;
      }


}/* Fin del if principal que efectua el Proceso de  REPOSICION */
if(hdr.WithDebug)
      userlog("[%s] ***** FIN    PROCESO DE REPOSICION ***** ", rqst->name );


/*********************************************************************
   A CONTINUACION SE LLAMA A LA FUNCION QUE REALIZA LA RECARGA,
   ESTE PROCESO SE REALIZA PARA CADA REGISTRO DE ENTRADA.
                  RECARGA
*********************************************************************/
if(hdr.WithDebug)
      userlog("[%s] ***** INICIO PROCESO DE RECARGA ***** ", rqst->name );

if( (nro_grupos > 0)  )
{
      ocurr=0;
      while(  Fpres32(fbfr, fld_IndCtaControlada, ocurr)   )
      {
```

```
        if(hdr.WithDebug)
            userlog("[%s]:  Recorriendo Buffer FML32 en proceso de RECARGA ocurrencia[%ld]", \
                                                                                rqst->name , ocurr );

        ci_fld_IndCtaControlada = ' ';
            if (( tux_getField(fbfr, fld_IndCtaControlada, ocurr, (char *)&ci_fld_IndCtaControlada,\
                                                                        &field_exists) ) == STATUS_FAIL)
            {
                if(field_exists == FIELD_NOT_EXISTS)
                {
                    userlog("[%s]:ADVERTENCIA: campo fld_IndCtaControlada  no fue recibido en msg. \
                                                        de entrada", rqst->name );
                    ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_IndCtaControlada no\
                                                        fue recibido en msg. de entrada" , &hdr );
                }
                else
                {
                    userlog("[%s]: ERROR en llamada a tux_getField(fld_IndCtaControlada)=[%d][%s]", \
                                                    rqst->name , Ferror32, Fstrerror32(Ferror32) );
                    ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror32(Ferror32) , &hdr );
                }
                goto _error;
            }
            else if(hdr.WithDebug)
                userlog("[%s]:     Se recibio fld_IndCtaControlada [%c]", rqst->name , ci_fld_IndCtaControlada );


            ci_fld_TipoDocCaja[0] = '\0';
            if (( tux_getField(fbfr, fld_TipoDocCaja, ocurr, ci_fld_TipoDocCaja, &field_exists) ) == STATUS_FAIL)
            {
                if(field_exists == FIELD_NOT_EXISTS)
                {
                    userlog("[%s]:ADVERTENCIA: campo fld_TipoDocCaja  no fue recibido en msg. de entrada",\
                                                                                    rqst->name );
                    ret = tux_setMsgHeader( HDR_LEVEL_WARNING , "campo fld_TipoDocCaja no fue recibido\
                                                                    en msg. de entrada" , &hdr );
                }
                else
                {
                    userlog("[%s]: ERROR en llamada a tux_getField(fld_TipoDocCaja)=[%d][%s]", rqst->name ,\
                                                                Ferror32, Fstrerror32(Ferror32) );
                    ret = tux_setMsgHeader( HDR_LEVEL_ERROR , Fstrerror32(Ferror32) , &hdr );
                }
                goto _error;
            }

            ret = rtrim(ci_fld_TipoDocCaja);

            if(hdr.WithDebug)
                userlog("[%s]:     Se recibio fld_TipoDocCaja [%s]", rqst->name , ci_fld_TipoDocCaja );


        if ( ci_fld_IndCtaControlada == '1' &&
                                    (
                                        strcmp( ci_fld_TipoDocCaja , TIPODOCCAJA_BOLETA ) == 0
                                      || strcmp( ci_fld_TipoDocCaja , TIPODOCCAJA_FACTURA) == 0
                                    )
           )
        {

            if(hdr.WithDebug)
                userlog("[%s]:              Se llamara a  call_Recarga ocurr[%ld] TipoDoc[%s]", rqst->name , ocurr,\
                                                                                    ci_fld_TipoDocCaja );
            if( call_Recarga( grupos,  fbfr , &hdr , ocurr , ci_fld_TipoDocCaja ) == STATUS_FAIL )
            {
                userlog("[%s]: Fallo llamada a call_Recarga" , rqst->name );
                goto _error;
            }

        } /* Fin del if que evalua el Ind. de Cta Controlada y si es Boleta o Factura. */
        ocurr++;

    }  /* Fin del while */
```

```
	} /* Fin del if principal que efectua el Proceso de  RECARGA */
	if(hdr.WithDebug)
		userlog("[%s] ***** FIN    PROCESO DE RECARGA ***** ", rqst->name );

	goto _exit;


_error:
	grp = grupos;
	while(grp)
	{
			if(hdr.WithDebug)
				userlog("[%s] saliendo por _error Eliminando nodo Cli[%s]Cta[%s]", rqst->name , grp->NroCliente, \
                                                                                   grp->NroCuenta );
		grupos = grupos->next;
		free((GRUPOS_CLIENTECUENTA *)grp);
		grp = grupos;
	}
	ret = tux_setDataError(rqst, hdr);
	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);

_exit:
	grp = grupos;
	while( grp )
	{
			if(hdr.WithDebug)
				userlog("[%s] saliendo por _exit Eliminando nodo Cli[%s]Cta[%s]TOcurrs[%d]BoF[%s]", \
                                rqst->name ,grp->NroCliente, grp->NroCuenta,grp->TotalOcurrencias, grp->BoF );
		grupos = grupos->next;
		free((GRUPOS_CLIENTECUENTA *)grp);
		grp = grupos;
	}

	ret = tux_setData(rqst, fbfr_aux,  hdr);
	if(fbfr_aux)
		tpfree((char *)fbfr_aux);
	if (ret == STATUS_FAIL)
		tpreturn(TPFAIL    , 0, rqst->data, 0L, 0);
	else
		tpreturn(TPSUCCESS , 0, rqst->data, 0L, 0);
}
```

```
/*
** Inicio del servidor
*/
int
#if defined(_STCD_) || defined(____cplusplus)
tpsvrinit(int argc,char **argv)
#else
tpsvrinit(argc,argv)
int argc;
char **argv;
#endif
{
     int  auth = 0;
     char strauth[20+1];

     if(ora_connect(argc,argv) == STATUS_FAIL)
          return(-1);

     /*
     ** ACTUALIZA EL NOMBRE DEL SERVIDOR
     */
     strcpy(GV_ServerName,argv[0]);
     if(  (auth=tpchkauth()) == -1 )
          return(-1);

     sprintf(strauth,"SECURITY_LEVEL=%d",auth);
     if(  tuxputenv(strauth) != 0)
          return(-1);


     /*
     ** Todo bien
     */
     return(0);
}

/*
** Termino del servidor
*/
void tpsvrdone()
{
     long ret;
     ret = ora_disconnect();
}
```

ANEXO No 4-9

CONFIGURACIÓN SERVIDOR GENÉRICO PARA RFC DE CAJA

```
# 1. R/3 login information
#CR3_DESTINATION=<R/3 System Name>
#CR3_CLIENT=<R/3 Client ID>
#CR3_USER=<R/3 User>
#CR3_PASSWORD=<R/3 Password>
#CR3_LANGUAGE=E
#SIDE_INFO=<your eLink app directory>\sideinfo
#CR3_EXIT_R3_CONNECT_LOSS=N
#CR3_TRACE=N

CR3_DESTINATION=SAPNODE
CR3_CLIENT=181
CR3_USER=elink
CR3_PASSWORD=bellprd
CR3_LANGUAGE=S
CR3_EXIT_R3_CONNECT_LOSS=N

SIDE_INFO=/home/mis/tuxsap/elink/sideinfo

CR3_TRACE=N
CR3_TRACE_FILE=/home/mis/tuxsap/elink/log/ctacte_cajas_tracefile.log

#
# Dado que algunas RFC retornan mas de 500 filas el buffer se fija en 512K
#
CR3_RESPONSE_BUFFER_SIZE=524288

# 2. Service names to be advertised
# 3. RFC names that services are mapped to
# 4. Import/export parameters and tables defined within the RFC
# 5. RFC row detail indicates structured parameters
#
#  SERVICE_LIST is limited in length to the number of bytes the
#    the environment variable can hold.  This is approximately
#    240 bytes.  Note that a service name is limited to 15 bytes
#    so we can have at most about 15 services.
#

SERVICE_LIST=Tx_jDocCtaCte

[SERVICE=Tx_jDocCtaCte]
CR3_RFC_NAME=Z_TX_CTACTECAJA
CR3_IMPORT_PARAMS=FLD_TIPOCONSULTACJ,FLD_SOCIEDAD,FLD_RUT,FLD_TIPODOCCAJA,FLD_FOLIODO
CUMENTO,FLD_CLIENTE,FLD_CUENTA,FLD_IDSAP
CR3_IMPORT_TABLES=
CR3_EXPORT_PARAMS=RETURN
CR3_EXPORT_TABLES=ZSALIDA
CR3_RFC_ROW_DETAIL=Y
```