SISTEMA SOPORTE DE DECISIONES GLOBAL DISTRIBUIDO

Pablo VILLARREAL, Alejandro TOFFOLO, Pablo ROSSI

Diseño y Desarrollo de Dominios: Griselda CASTRO, Leonardo GREGORET, Ma. Laura CALIUSCO, Lorena BEARZOTTI, Enrique MILANI, Federico WOSCOFF, Ma. Laura TAVERNA, Mariela RICO

MA. Rosa GALLI[**], Omar CHIOTTI[*]

GIDSATD-UTN – Facultad Regional Santa Fe – Lavaise 610- 3000 SANTA FE – Argentina
(*) **INGAR – CONICET** – Avellaneda 3657 – 3000 SANTA FE – Argentina
(**) **INTEC – CONICET** – Güemes 3450 – 3000 SANTA FE – Argentina
chiotti@arcride.edu.ar, mrgalli@intec.unl.edu.ar, pvillarr|prossi|jnagel|atoffolo@frsf.utn.edu.ar

RESUMEN

En este trabajo presentamos la arquitectura distribuida diseñada para un Sistemas Soporte de Decisiones Globales que está siendo desarrollado por el GIDSATD. En el diseño de esta arquitectura distribuida, el principal objetivo es reflejar la naturaleza de la aplicación (soporte a distintos puntos de decisión geográficamente distribuidos que deben cooperar entre sí para el intercambio de información) y lograr un bajo nivel de acoplamiento, de tal manera de obtener un sistema flexible y adaptable a los procesos de negocios de las organizaciones. El diseño propuesto se basa en una arquitectura distribuida federada y permite que la misma pueda ser construida con independencia del middleware usado en la comunicación de los componentes distribuidos. Para el diseño hemos usado el standard de Computación de Objetos Distribuidos CORBA junto con el Servicio de Eventos COSS OMG.

Palabras Claves

Sistema Soporte de Decisión, Sistema Distribuido, Arquitectura de Software, Interfaces Orientadas a Objetos

INTRODUCCIÓN

Según la nueva visión de las organizaciones, las empresas deben estructurarse alrededor de sus procesos productivos, organizando de manera eficaz todas las actividades que dan origen a la creación del valor agregado en los productos. El nuevo ambiente implica un cambio radical en el cual las tecnologías orientadas a procesar datos para convertirlos en información son fundamentales. En este contexto, conceptualmente se define a un Sistema Soporte de Decisiones (SSD) como una herramienta de ayuda al tomador de decisiones con el propósito de aumentar la efectividad y la eficiencia del mismo en el proceso de decisión. Muchos SSD han sido diseñados para servir como soporte de decisiones de determinados sectores de una organización (scheduling de personal, scheduling de producción, gestión y control de inventarios, planificación y control financiero, etc.), motivo por el cual la estructura de estos sistemas se ajusta a la situación particular a la cual sirven de soporte, y difícilmente puedan ser utilizables en otros esquemas de organización, ya sea de otras empresa o de la propia empresa.

Otro inconveniente, no menos importante, es que estos SSD fueron diseñados en forma independiente (no para trabajar en colaboración), de tal manera que la integración de varios de ellos en una organización para dar soporte a distintas áreas de decisión, si bien es viable mediante el uso de tecnologías de información emergentes como Middleware, generalmente se traduce en muy baja performance debido a las diferentes estructuras de datos, a la redundancia de los mismos, y a la superposición de funciones. Esta atomización del proceso de decisión, requiere de un adecuado sistema que brinde soporte a cada punto de decisión de la organización y que sea capaz de coordinar todo el proceso de decisiones. Hemos definido a estos sistemas como SSD globales [11]. Analizando los puntos de decisión de una organización es posible concluir que los mismos están ubicados en diferentes unidades funcionales dispersas geográficamente, que los modelos y técnicas de decisión son propios de cada tipo

de decisión y que cooperan entre sí intercambiando determinada información producto de las decisiones tomadas. Por lo tanto inferimos que un SSDG debería ser diseñado como un conjunto de subsistemas geográficamente distribuidos, operando con el menor nivel de acoplamiento posible, al cual denominamos Sistema Soporte de Decisiones Globales Distribuido (SSDGD). Esta descripción lleva al desarrollo de una aplicación inherentemente distribuida que debe estar reflejada por la arquitectura del SSDGD. El objetivo principal de este trabajo es obtener una arquitectura distribuida para el SSDGD donde el reflejo de la naturaleza de la aplicación y la búsqueda de un bajo nivel de acoplamiento se constituyan como principios de diseño, de tal manera de obtener un sistema flexible que se ajuste a la forma de trabajo de la organización, dando soporte a los procesos de negocios actuales de las organizaciones.

La estructura de la arquitectura estará conformada por un conjunto de componentes y debe diseñarse de una manera adecuada para poder llevar adelante las funciones de cada componente y la forma en que van a interactuar los componentes. A tal fin es conveniente utilizar tecnologías de orientación a objetos para poder definir interfaces orientada a objetos que definen la forma en que los componentes se relacionan entre sí. Otro aspecto importante que hace a la flexibilidad de una arquitectura distribuida es que debe ser diseñada independientemente de la tecnología de comunicación a utilizar.

En la primer parte del trabajo describimos brevemente diferentes estilos de arquitectura distribuida, y luego presentamos la arquitectura propuesta para el SSDGD. En la segunda parte describimos el diseño de la arquitectura distribuida del SSDGD a través del diseño cada uno de los componentes.

ESTILOS ARQUITECTÓNICOS PARA SISTEMAS DISTRIBUIDOS

En el desarrollo de un sistema de información distribuido la elección de la arquitectura a utilizar para su diseño depende de un conjunto de requerimientos funcionales y no funcionales que deben caracterizar al sistema de información distribuido. Existen varias arquitecturas de software para el desarrollo de un sistema distribuido, entre ellos, *Cliente/Servidor Two-Tier*, *Cliente/Servidor Three-Tier* y la *Arquitectura Federada.*

Por ejemplo, la *Arquitectura Federada* [14] propone un modelo uniforme de interacción que refleja el comportamiento de las organizaciones de hoy: varios procesos de negocios, varios focos de poder, grupos de usuarios que valoran su autonomía, y actividades geográficamente distribuidas posiblemente soportadas sobre plataformas heterogéneas. El modelo propuesto se denomina *Federación*, y está formado por una colección de *dominios* relativamente autónomos cooperando entre sí a través de un vinculo denominado *Federal Highway* al que están conectados. Este vínculo es el encargado de transportar los mensajes desde un dominio hacia otro. Un dominio es un grupo de aplicaciones altamente integradas que dan soporte a un sector de la empresa. El paradigma de comunicación empleado es el de Publicar/Suscribir y orientado al evento.

ARQUITECTURA DEL SSD GLOBAL DISTRIBUIDO

En la búsqueda de una arquitectura que refleje la forma natural de trabajo de los puntos de decisión y la manera en que se comunican entre ellos, hemos elegido construir la arquitectura del SSDGD en base a la Arquitectura Federada. Los componentes definidos en la arquitectura son los Dominios, los Gatekeepers y el Highway, los cuales se describen a continuación.Cabe destacar que, si bien generalmente se utiliza la Arquitectura Federada para la integración de sistemas de información existentes y heterogéneos donde cada sistema está constituido por un conjunto de aplicaciones altamente integradas, el objetivo de este trabajo es utilizar la Arquitectura Federada para la construcción y desarrollo de un nuevo sistema de información en todo su conjunto.

Dominios

Un dominio puede consistir en una aplicación o un conjunto de aplicaciones altamente integradas entre sí, que dan soporte a un sector determinado de la organización y las cuales se adaptan a la funcionalidad y forma de trabajo del área o departamento al cual pertenecen. Los dominios generalmente están geográficamente

distribuidos y poseen un alto nivel de autonomía, lo que permite que cada dominio pueda seguir el camino de desarrollo que crea conveniente, incorporando nuevas tecnologías, nuevas aplicaciones; todo tendiente a expandir su funcionalidad y responder a los cambios que se presenten dentro del área de la organización a la que pertenecen, sin que estos cambios afecten a otros dominios. La estructura y organización interna de un dominio está oculta para los demás dominios. Un dominio desconoce el funcionamiento de otros dominios.

Los dominios proveen información que necesitan otros dominios, como así también consumen información que reciben de otros dominios. Las responsabilidades y obligaciones que un dominio tiene en el SSDGD están definidas en un contrato. Un contrato especifica, entre otras cosas, los datos que se originan en el dominio y el formato relevante de tales datos, como así también los datos que son interés del dominio, y su formato relevante. Además un contrato especifica otras reglas adicionales respecto a las categorías de datos anteriores. Los elementos del contrato son incorporados dentro de varios componentes de software: dentro de las interfaces del dominio, dentro de las interfaces del Gatekeeper y en las colas de mensajes.

Las dependencias de procesamiento entre los dominios en la arquitectura del SSDGD están constituidas solamente por dependencias de información. Una dependencia de información [14] ocurre cuando como consecuencia de un evento dentro del dominio, éste necesita transmitir alguna información a dos o más dominios. La dependencia de información entre dominios induce a un más bajo acoplamiento y a un mayor grado de autonomía de los dominios. Sin embargo, dentro de un dominio pueden existir tanto dependencias de procesos, dependencias transaccionales o dependencias de información entre sus aplicaciones.

Los dominios definidos en la arquitectura del SSDGD se muestran en la figura 1 y definen los distintos puntos de decisión a los que va a dar soporte el SSDGD.

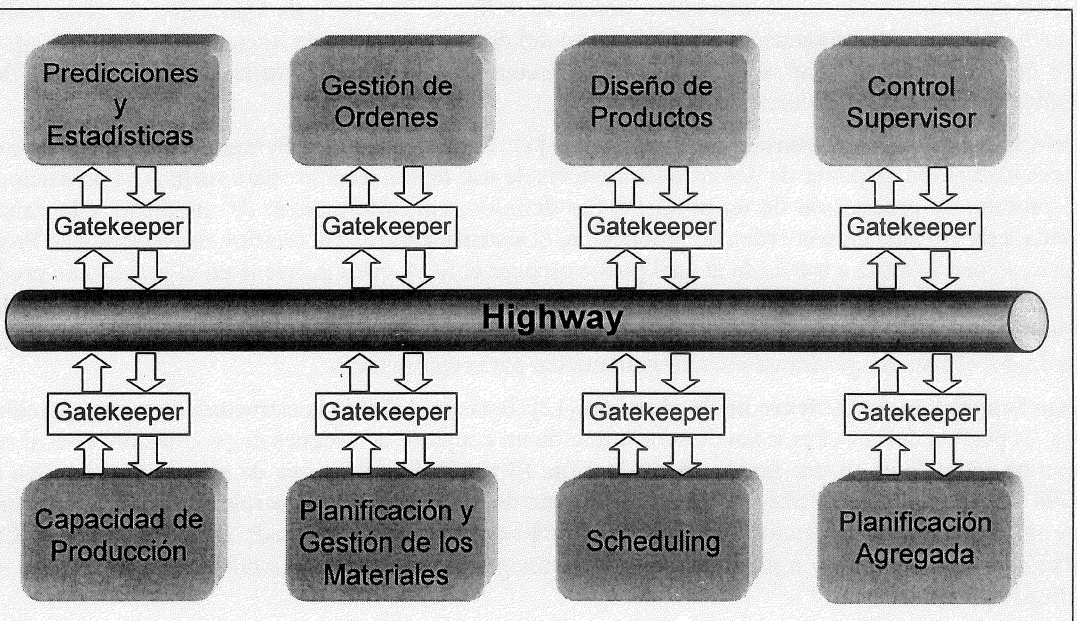

Figura 1: dominios del SSDGD

Breve Descripción de los Principales Dominios del SSDGD

Varios de los dominios que constituyen el SSDGD cuentan con un prototipo en funcionamiento. A continuación hacemos una breve descripción de algunos de ellos.

Dominio de predicciones y estadística [2]: El objetivo principal de este dominio es proveer un conjunto de herramientas que le permitan examinar el pasado y ayuden a predecir tendencias y eventos futuros. La base de modelos de este dominio incluye modelos de predicción y técnicas estadísticas que se requieren para brindar soporte al proceso de predicción. La arquitectura de este dominio es sufucientemente flexible y permite al experto incorporar sus propios criterios de valoraciones en las distintas etapas de un proceso de predicciones, como así también establecer el diseño de pronóstico que considere conveniente. La información aportada por este dominio es compartida por otros dominios tales como Planificación Agregada [4], Diseño de Producto [10] y Scheduling de Órdenes de Producción [12].

Dominio de planificación y gestión de materiales [3]**:** Este dominio utiliza el plan de producción estimado para realizar una estimación del cronograma de compra de materiales. Recibe del dominio soporte para la Planificación de Producción [4] el *plan de producción agregado actual* (expresado en término de familia de productos) y del dominio soporte de Predicciones [2] la predicción de demanda de productos terminados. Con esta información estima un p*lan de producción desagregado actual* a partir del cual, siguiendo la política de aprovisionamiento seleccionada por el usuario, el sistema genera las *órdenes estimadas de compra de materiales.*

Debe suministrar al dominio soporte para scheduling [12] información del cronograma de *disponibilidad estimada del material (k).* Esta información es generada a partir del i*nventario actual* del material (k), el cronograma de *órdenes de compra emitidas* del material (k) pendientes de ingreso, el cronograma de s*alidas comprometidas* del material (k) y el cronograma de ó*rdenes estimadas de compra* del material (k). Esta información define una restricción de disponibilidad del material (k) a ser tenida en cuenta por el scheduling.

Desde el punto de vista de la información necesaria para el scheduling, se considera la planta de producción como una entidad capaz de realizar actividades. La estrategia para representar el proceso de scheduling divide al mismo en dos niveles. Un nivel interactivo donde se define el Schedule de Operación de cada Actividad que constituye el proceso de producción (SOA) (a cargo del dominio Gestión de Recursos de Producción), y un nivel inferior que puede automatizarse, en el que el problema de scheduling es modelado (a cargo del dominio de Scheduling de Órdenes de Producción).

Dominio Gestión de los Recursos de Producción [1]**:** Para atender la solicitud de una orden de producción, cada actividad podrá disponer de distintas alternativas de uso de los recursos para realizar la prestación. Una vez que el proceso de producción de un producto fue definido, con información del producto y la cantidad a ser producida, especificada en una orden de producción, el sistema soporte a la Gestión de Recursos de Producción le da al usuario una interfaz a través de la cual puede escoger el SOA que interviene en el proceso de producción de este producto. El usuario podrá definir un único schedule relativo o podrá evaluar algunos schedules relativos que considere aceptables. Una vez que el SOA relativo de cada una de las actividades requeridas por el proceso ha sido definido, el sistema genera un modelo matemático para representarlo.

Dominio Scheduling de Órdenes de Producción [12]: Estando definida la estructura de representación del SOA relativo, es posible definir el proceso de scheduling de un conjunto de ordenes de producción, el cual requiere un conjunto de actividades de una planta de producción. El resultado de la tarea de scheduling implica decidir un Schedule de Actividades, un Shedule de requerimiento de Materiales, un Schedule de requerimiento de Mano de Obra y un Schedule de Servicios. Un Schedule está necesariamente asociado a un conjunto de Ordenes de Producción y ésta ,a su vez, a un Producto. Los cronogramas se generan usando un algoritmo de scheduling especificado por el usuario.

Dominio diseño de producto [10]**:** este dominio cuenta con una base de datos de los productos y sus posibles variantes así como una base de modelos para generar y evaluar diseños alternativos de productos respondiendo a las especificaciones del cliente. Los principales objetivos son: a) generar la estructura de un producto en línea; b) brindar soporte al cliente; c) dar soporte a la actividad de diseño de nuevos productos; d) evitar la redundancia de datos. Ligeras modificaciones en las características comerciales pueden implicar cambios en las actividades de producción cambiando los requerimientos de recursos, tiempo de operación, servicios mano de obra y costos. Toda esta información debiera estar disponible para el diseñador. Por esta razón, el dominio requiere en forma

explícita no solo la descripción comercial del producto, sino también la lista de materiales y el proceso de producción. La organización de la información asociada a las actividades que se desarrollan en la planta corresponde al dominio de Gestión de los Recursos de Producción, mientras que la información correspondiente a los materiales empleados en el diseño de un producto es responsabilidad del dominio de Planificación y Gestión de Materiales

Gatekeepers

Por cada dominio existe un Gatekeeper. Los Gatekeepers son componentes que interactúan con el dominio correspondiente y con los demás Gatekeepers. Cuando un dominio, como consecuencia de un evento, desea proveer información, su Gatekeeper asociado será el encargado de enviar esa información al Highway. Entonces los demás Gatekeepers tomarán esa información, si la misma es de interés para su dominio asociado. Por lo tanto, podemos definir a los Gatekeepers como los componentes de la arquitectura encargados de la ejecución de las acciones de publicar información hacia otros dominios y de consumir información desde otros dominios.

Como consecuencia, los dominios no interactúan explícitamente entre sí, sino que lo hacen a través de sus Gatekeepers. De esta manera también se libera a los dominios de tener que conocer y trabajar con los mecanismos del middleware que se elija para la comunicación. Son los Gatekeepers quienes van a usar el middleware para llevar adelante la comunicación por la red de información.

Highway

El Highway es el componente que nos permite abstraernos, en la arquitectura del SSDGD, de los detalles de comunicación de los componentes a través de las redes. El Highway estará constituido por el middleware adecuado para llevar adelante la comunicación por la red de información.

La decisión referente a qué middleware utilizar, para que los componentes de un sistema distribuido puedan interactuar, se ha vuelto una parte importante a la hora del diseño de un sistema distribuido. El tipo de middleware a utilizar debe proveer, además del mecanismo de comunicación subyacente a través de una red, un conjunto de servicios (tales como Naming, Marshalling de datos, Servicio de Eventos, comunicación Sincrónica/Asincrónica) que proporcionen un alto nivel de abstracción para que los diseñadores de la aplicación puedan concentrarse en los detalles de la aplicación y no en los detalles de comunicación de bajo nivel.

Uno de los objetivos del trabajo es definir una arquitectura que sea independiente del middleware a utilizar para posibilitar la comunicación entre los dominios. Para ello los Gatekeepers cumplen un papel importante como se verá en la etapa de diseño.

Esquema de Interacción entre los dominios

El modelo de comunicación en la arquitectura del SSDGD está basado en el paradigma de computación orientado a eventos (diferente a la computación por demanda sobre la cual se basan los sistemas Cliente/Servidor). Este modelo de comunicación permite que la interacción entre dominios sea iniciada por el dominio productor de una determinada información y además permite utilizar el paradigma de Publicar/ Suscribir. Los dominios, ante un evento, publicarán información que producen para que sea consumida por otros dominios suscriptos a esa información. El middleware de elección para la construcción del Highway debe dar soporte a estos mecanismos de comunicación.

Hemos identificado diferentes esquemas de interacción entre los dominios en la arquitectura del SSDGD que describen los procesos a los que da soporte el SSDGD dentro de una empresa [6], por ejemplo el proceso de Planificación Estratégica, que requiere la cooperación del dominio de Predicciones y Estadísticas con el dominio de Planificación Agregada; otro ejemplo de un proceso más complejo, el proceso de Scheduling, que requiere la interacción de varios dominios, entre ellos los dominios de Gestión de Ordenes, Scheduling, Diseño de Productos, Planificación y Gestión de Materiales, Capacidad de Producción y Control Supervisor.

Atributos de Calidad de la Arquitectura

De acuerdo a los requerimientos funcionales y no funcionales del SSDGD la arquitectura distribuida construida para el mismo satisface una serie de atributos de calidad que son importante para el SSDGD. Algunos de ellos se describen a continuación:

Escalabilidad: la arquitectura del SSDGD permite la incorporación de varios dominios sin afectar esto al funcionamiento de los dominios existentes. Más aún, los dominios pueden aumentar el número de aplicaciones, conformando estas aplicaciones fuertemente integradas alguna arquitectura distribuida, por ejemplo, podrían conformar una arquitectura Cliente/Servidor Three-tier dentro del dominio. Todos estos cambios pueden llevarse a cabo sin afectar a los restantes dominios de la arquitectura del SSDGD.

Modificabilidad: Los dominios pueden incrementar su funcionalidad sin afectar a otros dominios.

Integrabilidad: La arquitectura permite la incorporación de nuevos dominios que estén conformados por aplicaciones ya existentes y heterogéneas.

Portabilidad: Debido a la posible heterogeneidad entre los sistemas de los dominios y a la autonomía de los dominios, cada dominio puede operar sobre diferentes plataformas de hardware y software.

Reusabilidad: Todos los Gatekeepers son similares en sus interfaces y en su comportamiento, independientemente del funcionamiento del dominio al cual están asociados. Por lo tanto pueden ser reutilizados cuando se incorpora un nuevo dominio.

Adaptabilidad: Cuando los dominios deben proveer o consumir un nuevo tipo de información, sólo se requiere un cambio en las interfaces del dominio y del Gatekeeper, pero no se requiere un cambio en la funcionalidad del dominio. Tampoco estos cambios pueden afectar a los demás dominios.

DISEÑO DE LA ARQUITECTURA DEL SSDGD

En esta segunda parte del trabajo trataremos los aspectos de diseño de cada uno de los componentes de la arquitectura del SSDGD, empezando por el diseño de los dominios, de los gatekeepers y del highway.

Diseño de los Dominios

Actualmente los dominios están concebidos e implementados como aplicaciones independientes. De manera tal que nuestros dominios en la arquitectura del SSDGD están constituidos por una sola aplicación y no por un conjunto de aplicaciones fuertemente integradas.

Para poder cumplir con los atributos de calidad de la arquitectura, los dominios deben definir interfaces que indiquen como van a interactuar con los componentes de la arquitectura distribuida. Básicamente un dominio tiene que proveer información a otros dominios enviándola por el Highway como así también consumir información de otros dominios, pero el ejecutor de estas acciones es el Gatekeeper. Es decir que el dominio debe interactuar con el Gatekeeper, para cumplir con sus roles de proveedor y consumidor de información, a través de una interface bien definida.

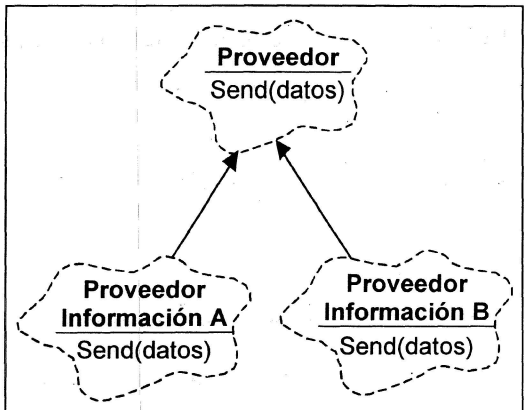

Figura 2. Interface en el Rol de Proveedor

Dominio en el Rol de Proveedor

Cuando el dominio actúa como proveedor de información la interface se define mediante un tipo de clase *Proveedor*. La idea conceptual de la clase *Proveedor* es la siguiente: permite proveer la información que es requerida por otros dominios del SSDGD. Es decir que permite proveer la información que el dominio publica. Cada dominio debe proveer distintos tipos de

información. Por lo tanto, por cada tipo de información que el dominio necesita proveer o publicar hacia otros dominios, se deriva una clase (por ejemplo una clase *Proveedor Información A*) desde el tipo de clase *Proveedor*, que proveerá el tipo de información de la que es responsable. El tipo de clase *Proveedor* y sus derivadas sólo tienen un método público llamado *Send (datos)*. Este método actúa como un procedimiento encargado de guardar la información a proveer a otros dominios, la cual está contenida en el parámetro datos, en una cola de mensajes correspondiente al tipo de información enviada, compartida con el Gatekeeper (En la próxima sección se ampliara sobre el concepto de cola de mensajes). La Figura 2 ilustra la estructura de la interface del dominio cuando éste actúa como Proveedor.

Dominio en el Rol de Consumidor

Cuando el dominio actúa como consumidor de información la interface se define mediante un tipo de clase *Consumidor*. La idea conceptual de la clase *Consumidor* es la siguiente: permite manejar la extracción de información necesaria para la operación del dominio que será consumida o utilizada por el mismo, y la cual es provista por otros dominios. Esta información se refiere a la información a la que está suscripto el dominio.

Al igual que cuando actúa como proveedor, cada dominio debe consumir distintos tipos de información. Por lo tanto, por cada tipo de información que el dominio necesita consumir desde otros dominios, se deriva una clase (por ejemplo una clase *Consumidor Información A*) desde el tipo de clase *Consumidor*, que extraerá el tipo de información de la que es responsable. El tipo de clase *Consumidor* y sus derivadas sólo tienen un método público llamado *Extract()*. Este método actúa como una función encargada de extraer la información a consumir desde una cola de mensajes correspondiente al tipo de información a consumir, compartida con el Gatekeeper. La función *Extract()* devuelve como resultado los *datos* que representan el tipo de información a consumir. La figura 3 ilustra la estructura de la interface del dominio cuando éste actúa como Consumidor.

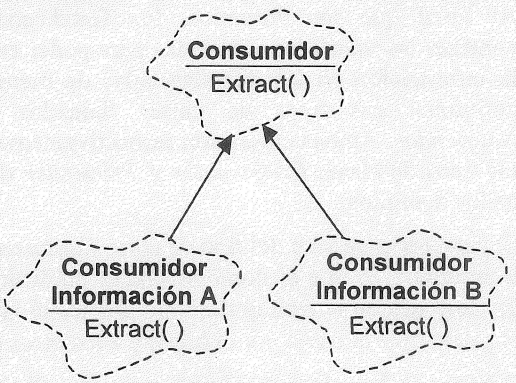

Figura 3. Interface en el Rol de Consumidor

Tipo de Información que proveen o consumen los dominios

Los *datos*, usados como parámetros del método *Send* del tipo de clase *Proveedor* y como resultados del método *Extract* del tipo de clase *Consumidor*, son objetos que representan la estructura completa de la información a proveer o a consumir. Estos objetos *datos* pueden ser creados y manejados de acuerdo a como el dominio lo crea conveniente. No obstante, estos objetos *datos* deben representar el tipo de información definida en los contratos de la arquitectura para que los dominios puedan entenderla.

Diseño de los Gatekeepers

Antes de comenzar con el desarrollo del diseño de los Gatekeepers, explicamos el concepto de cola de mensajes y su función dentro del diseño de la arquitectura.

Cola de Mensajes

Como se describió anteriormente, los dominios cuando actúan como proveedores guardan información en una cola de mensajes; cuando actúan como consumidores extraen información de una cola de mensajes.

Estas colas de mensajes permiten que el Gatekeeper sea el ejecutor del envío de la información que el dominio quiere publicar hacia otros dominios. También, permiten al Gatekeeper que la información proveniente de otros dominios sea guardada en la cola de mensajes para que posteriormente el dominio, en el rol consumidor, cuando esté disponible y en el tiempo que necesite esta información, la extraiga de la cola de mensajes. Por cada tipo de información que el dominio provee o consume se debe crear una cola de mensajes que almacene este tipo de información.

Gatekeeper

El Gatekeeper va a constituir otra aplicación independiente, que no forma parte de las aplicaciones de los dominios. Para que éste pueda desempeñar eficientemente sus funciones, debe operar en el mismo host donde también opera la aplicación que maneja la interface del dominio para con los Gatekeepers. También, como se verá posteriormente, el Gatekeeper es una aplicación que debe estar activa, ejecutándose siempre.

Cuando un dominio actúa en el rol de proveedor coloca la información a enviar en la cola de mensajes correspondiente al tipo de información a proveer. Luego, el Gatekeeper captura la información de esa cola de mensajes y la envía por el highway hacia otros dominios. Cuando un dominio actúa en el rol de consumidor extrae la información a consumir desde la cola de mensajes correspondiente. Anteriormente, el gatekeeper debió haber tomado esa información desde el highway y haberla almacenado en esa cola de mensajes, la cual se encuentra en un almacenamiento secundario.

El Gatekeeper es el responsable por la administración de las colas de mensajes que contienen la información a consumir o a proveer por el dominio. Es decir, además de extraer o guardar información en las colas de mensajes, debe limpiarlas cuando sea necesario y verificar que en todo momento estas colas estén en un estado consistente.

Al igual que los dominios, los Gatekeepers tienen que contener las interfaces adecuadas para poder extraer y guardar la información en las distintas colas de mensajes. Para ello definimos dos tipos de clases llamadas *Capturador* y *Notificador*. Ambas clases son respectivamente equivalentes a los tipos de clases *Consumidor* y *Proveedor* de las interfaces de los dominios.

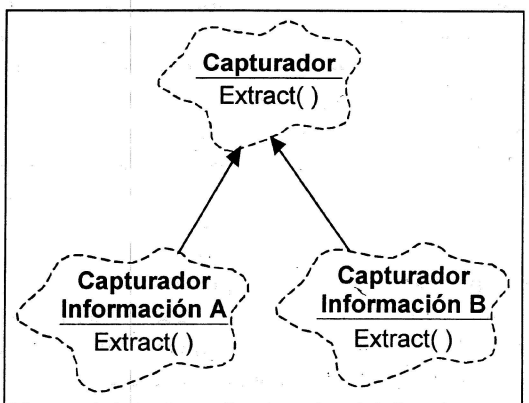

Figura 4. Interface *Capturador* del Gatekeeper

El gatekeeper se vale del tipo de clase *Capturador* para extraer la información que el dominio quiere publicar, desde la cola de mensajes en la cual el dominio almacenó la misma. Luego el gatekeeper la envía por el higway a los otros dominios.

Debido a que los dominios proveen distintos tipos de información y por cada tipo existe una cola de mensajes, se derivan del tipo de clase *Capturador* las clases que serán encargadas de extraer el tipo de información que está almacenado en la cola de mensajes que administran. Para ello, en la clase *Capturador* y en sus derivadas se define un método *Extract ( ): datos*, responsable de realizarlo. La estructura de estas clases se muestra en la figura 4.

Cuando un dominio actúa como consumidor de información, el gatekeeper se vale del tipo de clase *Notificador* para tomar la información que recibe desde el highway, perteneciente a otros dominios, y luego almacenar esta información en la cola de mensajes correspondiente desde la cual el dominio a través de su interface de consumidor la extraiga para su consumo. Este tipo de clase *Notificador* es la responsable de guardar la información que recibe desde el highway en la cola de mensajes correspondiente. Al igual que antes, por cada cola de mensajes, la cual almacena un tipo de información que el dominio desea consumir, se deriva de la clase abstracta *Notificador* las clases que serán responsables de almacenar el tipo de información que corresponda en la cola de mensajes que administran. Para ello, las clases *Notificador* y sus derivadas definen un método llamado *Send (datos)*, que realizará el almacenamiento de la información en la cola de mensajes adecuada. La estructura de estas clases se muestra en la figura 5.

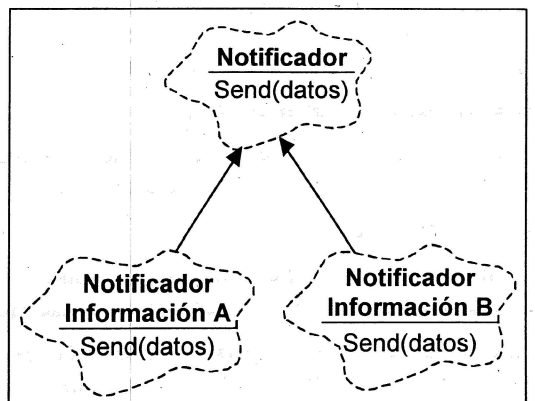

Figura 5. Interface Notificador del Gatekeeper

Debido a que los gatekeepers son los responsables de administrar las colas de mensajes, las clases *Capturador* y *Notificador* serán las encargadas de realizar la administración de las colas de mensajes definidas, además de sus funciones principales ya comentadas.

Para poder entender mejor la función y las interfaces del Gatekeeper podemos describirlo a éste como un módulo de software que independientemente de su estructura interna tiene cuatro interfaces: dos interfaces que manejan las colas de mensajes, definidas anteriormente con las clases *Capturador* y *Notificador*; una interface encargada de tomar la información que llega desde el highway publicada por otros dominios, y otra interface que se encarga de publicar la información del dominio para el cual trabaja el Gatekeeper, y pasarla al highway para que sea recibida por los otros dominios.

Interfaces para la comunicación entre Gatekeepers

A continuación se describirán las interfaces que permiten al Gatekeeper interactuar con el highway y con los demás Gatekeepers.

La comunicación entre los Gatekeepers se llevará a cabo a través del componente de la arquitectura que definimos como el Highway. El Higway será implementado con la tecnología de Computación de Objetos Distribuidos del standard CORBA del OMG, usando además el Servicio de Eventos de la Especificación de Servicios de Objetos Comunes del OMG (OMG COSS Event Service).

Para que los Gatekeepers puedan usar el Servicio de Eventos COSS OMG y CORBA deben definir dos tipos de clases que derivan de interfaces CORBA definidas en el Servicio de Eventos, y que constituirán las interfaces de cada Gatekeepers:

Una clase llamada *ProveedorGatekeeper*, responsable de enviar la información que el dominio desea publicar. Esta clase será derivada de la interface *PushSupplier*, la cual forma parte y está definida en el Servicio de Eventos.

Otra clase llamada *ConsumidorGatekeeper,* responsable de tomar la información que el dominio desea consumir, es decir la información a la que está suscripta el dominio. Esta clase será derivada de la interface *PushConsumer*, la cual forma parte y está definida en el Servicio de Eventos.

A continuación se muestra como realizan sus funciones estas interfaces y se describe el diseño del Highway.

Diseño del Highway

Los Gatekeepers deben comunicarse con otros Gatekeepers para que los dominios intercambien información. Para ello requieren de una infraestructura de comunicación apropiada. Uno de los requerimientos de la arquitectura, es que ésta fuera independiente del middleware a utilizar. Ahora podemos afirmar que esto se cumple ya que son los Gatekeepers los que van a manejar los detalles de cómo utilizar el middleware elegido. De esta manera podríamos utilizar cualquier middleware (por ejemplo: CORBA, COM, MOM o Java RMI) porque los dominios no van a trabajar con uno en particular sino que esto lo va a ser el Gatekeeper correspondiente. Más aún, si en un momento se estuviera utilizando un tipo de middleware en particular y se reemplaza por otro, el cambio no afectará a los dominios. Ni siquiera se va a necesitar cambiar las interfaces *Capturador* y *Notificador* del Gatekeeper. Sólo se deberían cambiar las interfaces *ProveedorGatekeeper* y *ConsumidorGatekeeper* que son las que se comportarán de acuerdo al middleware elegido.

Como se mencionó anteriormente, se decidió utilizar como middleware al standard CORBA del OMG [8] que posibilita la Computación de Objetos Distribuidos. Como Servicio CORBA esencial se utiliza el Servicio de Eventos COSS OMG [9], además del Servicio de Nombres.

Descripción del Servicio de Eventos COSS OMG

El modelo de comunicación de la arquitectura del SSDGD requiere que los gatekeepers utilicen un paradigma de comunicación asincrónico, desaclopado y multicast. El Servicio de Eventos COSS OMG, basado en el standard CORBA, permite dar soporte a un paradigma de comunicación asincrónico (el transmisor de un mensaje no espera una respuesta desde el receptor luego del envío del mismo), desacoplado (los objetos y sus clientes se

comunican a través de un intermediario, y de esta manera no necesitan tener conocimiento uno de otro) y multicast (el mismo mensaje generado por un objeto se envía a múltiples clientes) entre objetos distribuidos.

El Servicio de Eventos COSS OMG define tres componentes primarios:

Proveedores y Consumidores: Los *Proveedores* producen datos y los *Consumidores* procesan los datos. Los consumidores son los objetivos de los datos generados por los Proveedores. Los Proveedores y los Consumidores pueden jugar roles activos o pasivos [9].

Canal de Eventos: La abstracción central en el Servicio de Eventos es el *Canal de Eventos*, el cual es un objeto interviniente que permite a múltiples proveedores comunicarse con múltiples consumidores en forma desacoplada y asincrónica. Un Canal de Eventos es un consumidor y un proveedor de datos. Los Canales de Eventos son objetos CORBA standard y la comunicación con un Canal de Eventos es llevada a cabo usando peticiones CORBA standard. El Canal de Eventos es un "mediador" entre Consumidores y Proveedores. Aparece como un "proxy" del Consumidor a los Proveedores reales por un lado y como un "proxy" del Proveedor a los Consumidores reales por el otro.

Modelos de comunicación del Servicio de Eventos

Existen cuatro modelos generales de colaboración en la arquitectura del Servicio de Eventos COSS OMG [13]. Para el diseño del highway en la arquitectura del SSDGD se usará el Modelo de Comunicación Push Canónico. Este modelo, mostrado en la Figura 6, permite a los Proveedores de datos iniciar la transferencia de datos a los Consumidores. Los Proveedores son los iniciadores activos y los Consumidores son los objetivos pasivos de las peticiones *push()*. Los Canales de Eventos juegan el rol de *Notificador*, como se define en el patrón Observer [5]. De esta manera los Proveedores activos usan los Canales de Eventos para poner datos a los Consumidores pasivos que se han registrado con el Canal de Eventos.

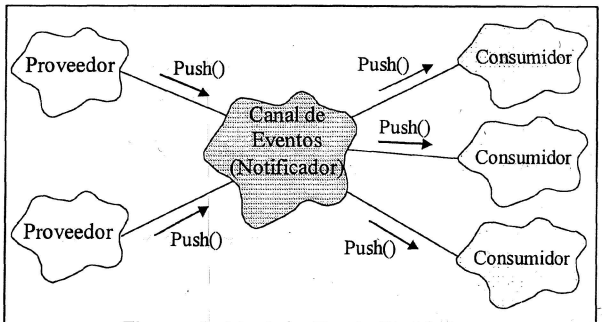

Figura 6. Modelo Push Canónico

El Servicio de Eventos define las interfaces que permiten que estos objetos interactúen. Las interfaces del Servicio de Eventos están definidas usando el Lenguaje de Definición de Interfaces del OMG (IDL OMG). Existen dos interfaces que son importantes para describir el funcionamiento del highway que utilizará el Modelo Push Canónico en el Servicio de Eventos. Estas son: la interface *PushSupplier*, para el proveedor, y la interface *PushConsumer*, para el consumidor. Además, existen otras interfaces, como la interfaces *EventChannel* del Canal de Evento y varias otras que permiten conectar los proveedores y consumidores con el Canal de Eventos. A continuación se muestran las interfaces *PushSupplier* y *PushConsumer* en IDL OMG:

```
interface PushConsumer {
   void push(in any data) raises (Disconnected);
   void disconnect_push_consumer();
};
interface PushSupplier {
   void disconnect_push_supplier();
};
```

El Servicio de Eventos en la Arquitectura del SSDGD

Cada Gatekeeper debe implementar tanto la interface *PushSupplier* como la interface *PushConsumer*. Por lo tanto cada uno, debe definir una clase *ProveedorGatekeeper* derivada de la interface *PushSupplier* y otra clase *ConsumidorGatekeeper* derivada de la interface *PushConsumer*. Estas clases deben implementar los métodos *disconnect_push_supplier()* y *disconnect_push_consumer()* para realizar las acciones adecuadas cuando el Canal de Eventos quiere desconectarlos. La clase *ConsumidorGatekeeper* en un Gatekeeper también debe implementar

el método *push*, el cual será invocado por el Canal de Eventos cuando otro Gatekeeper emite una petición *push*. Éste método le permite al objeto *ConsumidorGatekeeper* obtener la información que otro dominio le está enviando la cual estará contenida en el parámetro data y será luego almacenada por el objeto *Notificador* en la cola de mensajes correspondiente para que luego el dominio actuando en el rol de consumidor la pueda consumir.

Funcionamiento de los Gatekeepers con el Servicio de Eventos

Antes de establecer la comunicación entre los Gatekeepers, se debe "levantar" en algún servidor o host de la red, un objeto CORBA correspondiente al Canal de Eventos. Este objeto debe estar siempre "levantado" para poder realizar la comunicación entre los objetos *ProveedorGatekeeper* y *ConsumidorGatekeeper* de los Gatekeepers.

Cuando un Gatekeeper comienza a ejecutarse debe instanciar los objetos *ConsumidorGatekeeper* y *ProveedorGatekeeper y* realizar los pasos correspondientes para conectar estos objetos al Canal de Eventos usando los métodos de las interfaces del Servicio de Eventos.

Los Gatekeepers también deben estar siempre "levantados" para poder recibir en todo momento peticiones *push* para el objeto *ConsumidorGatekeeper*. El Gatekeeper pasa mucho de su tiempo en un loop esperando que arriben datos desde otros Gatekeepers, es decir, esperando por peticiones *push*. Además deben estar "corriendo" para que cada cierto tiempo los objetos *Capturadores* revisen en sus colas de mensajes si tienen información para extraer.

El objeto *ConsumidorGatekeeper* recibe una invocación al método *push(datos)*, toma el parámetro *datos*. y determina si los datos son del tipo de información definido en el contrato correspondiente al dominio. Si es así, lo convierte al formato adecuado si es necesario, se instancia el objeto *Notificador* que corresponda y se invoca el método *send(datos)* para enviar los datos que recibió el Gatekeeper a la cola de mensajes, para luego ser consumida por el dominio. Si no recibe el tipo de información a la que está suscripto, simplemente la ignora.

Cuando un objeto *Capturador* de un Gatekeeper revisa la cola de mensajes correspondiente y encuentra una nueva información que el dominio quiere proveer a otros dominios, se invoca el método *extract* del objeto capturador quien extrae los datos de la cola de mensajes. Luego se realiza la conversión de los datos, si es necesario, y se invoca el método *push(datos)* a un objeto proxy dentro del Canal de Eventos. Posteriormente, el Canal de Eventos invoca el método *push* de todos los objetos *GatekeeperConsumidores* que estén conectados al Canal de Eventos, en los Gatekeepers activos.

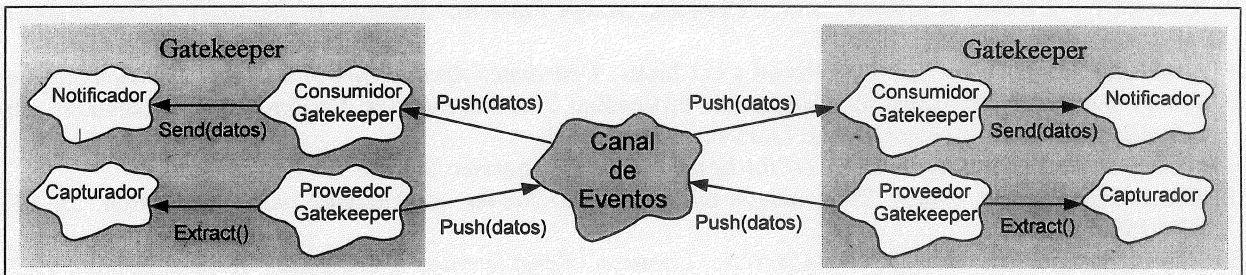

Figura 7. Comunicación entre dos de los Gatekeepers usando el Servicio de Eventos

CONCLUSIONES

La arquitectura del SSDGD descripta ha permitido cumplir con el objetivo de posibilitar que los dominios, que dan soporte a los distintos puntos de decisión de una organización, trabajen y cooperen entre sí de acuerdo a la forma natural de trabajo de los sectores o departamentos de una organización, brindándole a los mismos un alto nivel de autonomía. La arquitectura del SSDGD posibilita que las aplicaciones que conforman un dominio se relacionen con un bajo nivel de acoplamiento.

La arquitectura utiliza un modelo de comunicación orientado a eventos, con el paradigma Publicar/Suscribir, que posibilita que el SSDGD dé soporte a la forma de comunicación entre los dominios para que todos ellos se adapten a las necesidades de los procesos de negocios de las organizaciones.

La arquitectura del SSDGD, derivada de la Arquitectura Federada, cumple con distintos atributos de calidad importantes para un SSD y para un sistema distribuido, alcanzando un alto nivel de flexibilidad para seguir en el futuro con el desarrollo de nuevas aplicaciones, que constituirán nuevos dominios a integrarse al SSDGD, y que darán soporte a otros puntos de decisión de las organizaciones.

Las interfaces orientadas a objetos definidas para cada uno de los componentes posibilitó construir la forma en que los distintos componentes interactúan entre sí en la arquitectura del SSDGD.

La elección de CORBA con el servicio de eventos COSS OMG para la construcción del componente highway posibilitó abstraer en el diseño los detalles de comunicación de bajo nivel y por otro lado, más importante aún, posibilitó construir el modelo de comunicación Publicar/Suscribir orientado a evento que toda Arquitectura Federada debe cumplir. Actualmente se está considerando utilizar un nuevo servicio de eventos llamado Servicio de Notificación [7], el cual está basado en el anterior y que posibilitará adicionar nuevas capacidades como la entrega confiable de mensajes, el filtrado de datos, la interconexión de varios canales de eventos, repositorio de eventos, calidad de servicio y la mejora de la performance en la red.

REFERENCIAS

[1] Azcoaga J., Pletchuck V., Luciani G., Busaniche F. Y Chiotti O. *Global Decision Support System Generator for Industrial Companies: Support System of Resource Management.* Procceding 25th International Conference on Computers and Industrial Engineering. New Orleans; Lousiana, Marzo 1999

[2]Caliusco M.L., P.Villarreal, A.Toffolo, M.L.Taverna and O.Chiotti, *Decision support system generator for industrial companies. Module IV: forecasting support system.* Computer & Industrial Engineering, Vol. 35, Nos 1-2, pp315-318 (1998)

[3] Castro G., L.Gregoret, W. Delfabro, O. Chiotti., *Decision support system generator for industrial companies. Module II: System to support the material mangement* . Computer & Industrial Engineering, Vol. 35, Nos 1-2, pp 307-310 (1998)

[4] Contin C., M.A.Gomez,, M.R.Galli y O.Chiotti *DSS Generator for Aggregate Production Planning.* Proceeding 25th Int. Conference on Computers and Ind. Engng. New Orleans; Lousiana, Marzo 1999.

[5] Gamma E., R.Helm, R.Johnson, and J.Vlissides, *Design Patterns: Elements of Reusabel Object-Oriented Software*, Addison-Wesley, 1995.

[6] Nagel J., P.Rossi, A.Toffolo, P.Villarreal y O.Chiotti . *Distributed Architecture of a Global Decision Support System Generator for Industrial Companies*. Proceeding 25th International Conference on Computers and Industrial Engineering. New Orleans; Lousiana, Marzo 1999.

[7] NEC Systems Laboratory, Inc., *A CORBA-based Notification Service*, Agosto 1997.

[8] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.0 ed., Julio 1995.

[9] Object Management Group, *CORBAServices: Common Object Services Specification, Revised Edition*, 95-3-31 ed., Marzo 1995.

[10]Ramos,J.C., Bearzotti, L., E. Milani, , F. Woscoff, M. Gorsky, M. R. Galli, O. Chiotti, *DSS Generator for Industrial Companies. Module I: Product Design Support System*, Computers and Industrial Engineering, Vol. 35, Nos. 1-2,pp 303-306 (1998)

[11] Rico M., O.Yuschak, M.L. Taverna, J.Ramos, M.R. Galli y O. Chiotti, *Decision Support Systems Generator for Industrial Companies,* Computers and Industrial Engineering, Vol 33, 1-2, pp. 357,360, 1997.

[12]Rossi, P., C.Díaz, P. Fruttero, C.Vittori and O.Chiotti. *DSS Generator for Industrial Companies. Module III: Scheduling Support System.* Computers and Industrial Engineering, Vol.35, N°s 1-2, pp. 311-314. (1998)

[13]Schmidt D. C. y S. Vinoski, *The OMG Events Service (Column 9), C++ Report, vol. 9*, Febrero 1997.

[14]Wijegunaratne I. and G. Fernández, *Distributed Applications Engineering*, Springler, 1998.