UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMATICA

TESIS DE GRADO

ARQUITECTURA DE UN BUS DE SERVICIO EMPRESARIAL

PARA OPTAR AL TITULO DE LICENCIATURA EN INFORMATICA

MENCION: INGENIERIA DE SISTEMAS INFORMATICOS

POSTULANTE: RUBEN DARIO CHALCO CAMPOS

TUTOR: Lic. EFRAIN SILVA SANCHEZ

REVISOR: Lic. JUAN CONTRERAS CANDIA

LA PAZ – BOLIVIA

2011

DEDICATORIA

A mi madre una mujer incansable en el tiempo.

A mi padre un hombre que es la prueba de fuerza de voluntad viviente.

A mi familia y a todo aquel que fue, es y será parte de esta casa superior de estudios

AGRADECIMIENTOS

A Lic. Efrain Silva por su dedicación en la labor de formar nuevos profesionales en nuestra sociedad.

A Lic. Juan Contreras por su dedicación en la revisión del presente trabajo.

A Dios, a mi patria y a mi hogar.

RESUMEN

La concepción postmoderna de la construcción de software dio lugar al nacimiento de distintos conceptos como arquitectura de software, nuevos paradigmas como SOA y finalmente la cima de esta era ESA (Aplicativos de Servicios Empresariales), sin embargo un tema controversial que nace es la integración de los procesos comerciales con su contraparte tecnológica los Servicios de los aplicativos. El presente trabajo pretende formalizar esta relación contraponiendo un concepto base "Bus de Servicio Empresarial".

En el primer capítulo veremos cómo fue evolucionando la construcción de software desde aplicativos monolíticos hasta la arquitectura orientada a servicios, posteriormente definiremos conceptos que aparecieron a mediados de la década del 2000.

Durante el segundo capítulo revisaremos conceptos importantes para la investigación como la arquitectura, los patrones de diseño empresarial y la definición de una aplicación empresarial.

En el tercer capítulo diseñaremos la especificación arquitectónica del Bus de servicio empresarial, tomando como contexto inicial las necesidades que priman en la integración y gestión de los servicios, definiremos la construcción de los componentes asentados en un framework base WCF.

Finalmente describiremos los pros y contras de usar un ESB como herramienta dentro del flujo comercial de una organización.

La percepción que toma la presente investigación es de tesis explicativa sobre un campo de estudio "Ecosistemas de aplicativos caóticos", contemplando una serie de factores que permiten convertir este en un ambiente totalmente ordenado y flexible.

INDICE

CAPITULO I

INTRODUCCION

I. INTRODUCCION

Actualmente los procesos de producción y gestión de infinidad de organizaciones son automatizados, esto con el fin de alcanzar altos niveles de eficiencia y eficacia en los mismos, para poder cumplir las expectativas del mercado actual; consecuencia de este hecho es la concepción de distintos tipos de sistemas de información que pueden clasificarse en su mayoría como servicios empresariales (ej. ERP, CRM, SCM, etc.), cada uno con una estructura y comportamiento interno particular - a la cual denominaremos arquitectura posteriormente - misma que será el objeto de nuestro estudio.

Cada servicio empresarial que compone este flujo comercial se considera un beneficio para la institución y como parte de la misma infiere en la gestión de sus políticas, mismos que consideran temas como la integración de sistemas, mismo que este en función al flujo comercial. Definiremos "flujo comercial" como la secuencia de pasos que sigue los procesos de una empresa en su cadena de producción, por ejemplo el inicio de registro de recursos insumos, manejo de almacén, contabilización de movimientos, registro de la planilla de funcionarios, etc. por mencionar alguno de los procesos.

Cada uno de estos procesos o servicios internos de la empresa está compuesto de una serie de pasos, por ejemplo la contabilidad inicia con la identificación de cada uno de los movimientos, la generación de asientos contables, la generación de libro diario, generación de reportes analíticos de cierres mensuales, etc. A toda esta secuencia de pasos la denominaremos "Lógica Comercial", mucho se ha hablado de la misma, una de sus características principales es el cambio constante que sufre, motivo por el cual se tiene que realizar nuevas adecuaciones a los sistemas de información y/o aplicaciones que son los componentes responsables de realizar estas tareas.

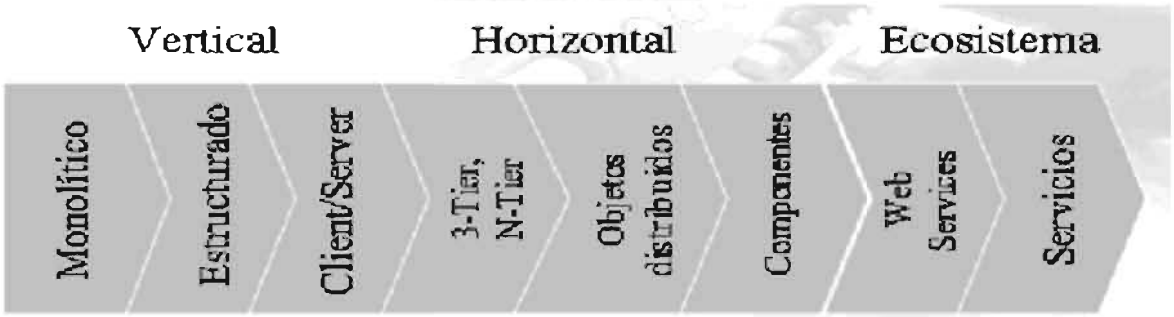
Considerando que se tienen un conjunto de servicios, cada uno compuestos a sus vez de una serie de sistemas de información, la gestión del flujo comercial es una tarea compleja, ya que el control de esta no solo implica controlar las tareas criticas de cada uno de los servicios sino también evaluar el impacto de los posibles cambios del flujo comercial en todos los sistemas de información, comúnmente estas tareas de gestión tienen costos altos en la empresa; ya que muchas veces no se cuentan con las herramientas adecuadas para poder medir tanto el impacto de estos cambios.

En la actualidad se han realizados varios estudios sobre el comportamiento de las aplicaciones empresariales, desde su arquitectura hasta los patrones que siguen estas. Uno de los temas más controversiales es la integración de los sistemas de información, el cómo realizar esta tarea con éxito no solo en el buen funcionamiento de los aplicativos si no también con la optimización del flujo comercial, ha concebido el concepto de "Bus de Servicios Empresariales" objeto de estudio presente trabajo.

1. ANTECEDENTES

Como parte de la evolución tecnológica de los sistemas de Información se adoptaron nuevos comportamientos arquitectónicos, uno de ellos es SOA, que logra integrar los recursos humanos e información en un ambiente más operativo y flexible, centrándose en el estudio de los procesos de negocios de una organización.

El siguiente grafico muestra de forma cronológica las distintas etapas por las cuales diversos modelos de arquitectura fueron aplicados en la construcción de sistemas de información:



En la etapa de Ecosistema – sistema compuesto de organismos relacionados entre sí que comparten un mismo habitad – vemos la existencia de un elemento ***Servicio***, concepto base de SOA (Arquitectura Orientada a Servicios), algunas definiciones de este modelo son:

"La Arquitectura SOA establece un marco de diseño para la integración de aplicaciones independientes de manera que desde la red pueda accederse a sus funcionalidades, las cuales se ofrecen como servicios. La forma más habitual de implementarla es mediante Servicios Web, una tecnología basada en estándares e independiente de la plataforma, con la que SOA puede descomponer aplicaciones monolíticas en un conjunto de servicios e implementar esta funcionalidad en forma modular"

Microsoft Corporation

Publicado: Diciembre 2006

"Arquitectura Orientada a Servicios (SOA), es un marco conceptual para integrar procesos de negocios soportados en tecnología segura a través de componentes desarrollados bajo estándares internacionales que pueden ser re-utilizados y combinados para adaptarse a los cambios de prioridad del negocio"

SIS SOAction

"La Arquitectura Orientada a Servicios de cliente (en inglés Service Oriented Architecture), es un concepto de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio.

Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros"

Wikipedia

Para el contexto en el cual deseamos incurrir definiremos SOA como:

"Marco de diseño arquitectónico basado en unidades lógicas llamados servicios, mismos que son parte de un ecosistema en donde el habitad son los procesos de negocio"

Definiremos a ***Servicio*** como un componente SOA, que encapsula la funcionalidad de un proceso de negocio, se detallan sus cualidades:

1. **Los Servicios deben ser reusables**: Todo servicio debe ser diseñado y construido pensando en su reutilización dentro de la misma aplicación, dentro del dominio de aplicaciones de la empresa o incluso dentro del dominio público para su uso masivo.
2. **Los Servicios deben proporcionar un contrato formal**: Todo servicio desarrollado, debe proporcionar un contrato en el cual figuren: el *nombre del servicio, forma de acceso, las funcionales que ofrece, los datos de entrada de cada una de las funcionalidades y los datos de salida.* De esta manera, todo consumidor del servicio, accederá a este mediante el contrato, logrando así la independencia entre el consumidor y la implementación del propio servicio. En el caso de los Servicios Web, esto se logrará mediante la definición de interfaces con WSDL (Web Services Description Language).
3. **Los Servicios deben tener bajo acoplamiento**: los servicios tienen que ser independientes los unos de los otros; esto es posible siempre y cuando un servicio sea accedido a través de un contrato, simulando un comportamiento de caja negra.
4. **Los Servicios deben permitir la composición**: Todo servicio debe ser construido de tal manera que pueda ser utilizado para construir servicios genéricos de más alto nivel, el cual estará compuesto de servicios de más bajo nivel.
5. **Los Servicios deben de ser autónomos**: Todo Servicio debe tener su propio entorno de ejecución y representar algún proceso de negocio.
6. **Los Servicios no deben tener estado**: Un servicio no debe guardar ningún tipo de información, un servicio es un contenedor lógico, no obedece a un patrón de *Workflow*.
7. **Los Servicios deben poder ser descubiertos**: Todo servicio debe poder ser descubierto mediante uno o más protocolos de comunicación. En el caso de los Servicios Web, el descubrimiento se logrará publicando los interfaces de los servicios en registros UDDI. (Universal Description, Discovery, and Integration).

Ventajas de implementar SOA

Podemos encontrar varias ventajas al implementar servicios empresariales orientados a SOA, mencionemos alguno de ellos:

- Conocimiento exacto del modelo del negocio actual.
- Flexibilidad al cambio, por lo tanto mayores posibilidades de innovación empresarial.
- Desmitificación de las especificaciones arquitecturales de los servicios en una empresa.
- Independencia de la tecnología de desarrollo.
- Centralización de la gestión de la información.
- Gestión de infraestructura granular.
- Estandarización sin sacrificar la flexibilidad.
- Reducción de costos de operación, con la reutilización de servicios.
- Agilidad y productividad.

Y se podrían mencionar aun varias más, sin embargo una implementación incorrecta este paradigma arquitectónico podría incurrir en varias desventajas y problemas para la institución, mismo que analizaremos en la siguiente sección del presente documento.

Conceptos posteriores a SOA

Con la aparición de SOA como modelo basado en servicios mismos previamente descritos, también crecen los conceptos en IT y en la generación de procesos de negocio como: Servicios Empresariales, la comunicación inteligente mediante el envío de mensajes entre sistemas de información que al final determina en el nacimiento de los conceptos EAI y ESA:

EAI - Enterprise Application Integration (Integración de Aplicativos Empresariales)

Irónicamente este concepto se concibe en el entorno de la división de aplicaciones, en si la división de los componentes de software dentro de uno o varios sistema de información y las formas de integración que uno puede optar.

Varias veces encontramos aplicativos que tiene la siguiente forma: una capa de acceso a datos, una capa de entidades de negocio y una capa para la interfaz del usuario, esta última generalmente acumula un comportamiento específico de las reglas de negocio, a medida que van aumentando las interfaces, se hace más duro tener una buena gestión de políticas del negocio desde el punto de vista de tecnológico.

Para lo mismo EAI propone estas buenas prácticas:

- *Externalizar el área de intercambio de información entre aplicaciones*: teniendo encapsulada las reglas de negocio en una capa, se puede transmitir mensajes entre las mismas, estandarizando los contratos de mensaje enviados. La ventaja principal indica la separación de la lógica del servicio, de la lógica del flujo comercial de la Empresa.

- *Externalizar el área de persistencia de datos de las aplicaciones*: la reutilización de una misma capa de acceso a datos permite la independizarían de la interfaz de usuario (incluyendo los posibles dispositivos por los cuales se manifestara la interfaz) del proveedor de datos. Sin embargo las ventajas principales de tener una fuente de datos compartida son:
    - *Sincronización de la información en tiempo real.*
    - *Disponibilidad de hacer mejores estudio de los datos.*
    - *Facilidad de recuperación en posibles incidentes.*
    - *Manejo estándar de la metada de los repositorios*
- Encapsulación de los servicios en un Área virtual de negocios: se trata de tener un área para alojar servicios empresariales como un BMP, ERP, CMR, etc. Aventajándonos mediante:
    - Una fácil definición y mantenimiento de los procesos de negocio.
    - Monitoreo en línea de estado de los procesos de negocio.

ESA Enterprise Services Application (Aplicaciones de servicio empresarial).

Definiremos a ESA como la vista simple de aplicaciones complejas, si bien SOA trae tantos beneficios aun deja muchas preguntas sin contestar por ejemplo cómo se forjarán las partes reusables, quien las construirá, lo que las herramientas serán usadas. Aquí hay simplemente algunas preguntas que deben ser contestadas para derivar el valor comercial completo de aplicaciones complejas:

¿Cómo deberían ser los componentes de software a través de esta arquitectura?

¿Cómo deberían comunicarse dos componentes?

¿Cuál es la estructura correcta para cada componente?

¿Cómo deberían construirse las interfaces de usuario?

¿Cómo debería tener lugar la orquestación de proceso?

¿Dónde está la persistencia?

¿Cómo se manejan los datos distribuidos?

¿Cómo puede ser manejada la complejidad?

¿Cómo puede adaptar un proceso de aplicaciones simplificado?

¿Cómo pueden ser los desarrolladores más productivos?

¿Cuál es la división del trabajo correcta?

¿Quién debería solucionar todo estos problemas?

Para comenzar, miremos algunas áreas diferentes más de cerca. Por ejemplo, en una aplicación compleja, los datos son distribuidos sobre varios repositorios. ¿Cómo una versión unificada puede ser ensamblada? ¿Cómo pueden afectar los cambios en los registros en muchos repositorios sincronizados?

Existen varios tipos de formas diferentes de lógica de proceso. Hay un flujo de trabajo, lo cual ocurre dentro de una aplicación, hay orquestación en un proceso, lo cual ocurre dentro de una aplicación compleja y está la lógica que es manejada por un proceso de aplicación empresarial a la

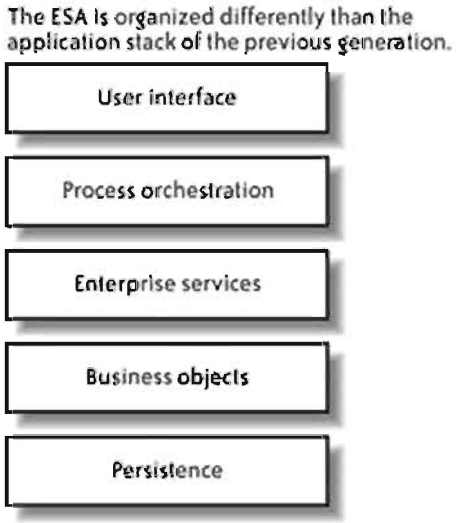
cual llamaremos lógica de integración de proceso. ¿Cómo construir los componentes para que la lógica del proceso de integración, la lógica aplicativa, etcétera, además bajo la condición de que los repositorios son distribuidos? ¿Toda la lógica estará en medio? ¿Las aplicaciones de la empresa tendrán que volverse más pequeñas? ¿Cómo deberían ser las aplicaciones desarrolladas? ¿Se necesitan los métodos nuevos de desarrollo? ¿Cómo pueden ser las cosas simplificadas? ¿Quién va para hacer el trabajo de crear los servicios que se edificó sobre las aplicaciones de la empresa? ¿Hay algún estándar para esos?¿Cómo las normas de industria serán incorporadas? ¿Cómo participar a todos de la arquitectura y proceso del diseño en curso?

Es realmente claro que queda aun muchísimas preguntas. Uno, sin embargo, es quizá lo más fundamental: ¿Quién debería contestar las preguntas? ¿Cada DEPARTAMENTO DE INFORMÁTICA?

Uno de los aspectos más importantes de ESA es la fórmula que tiene previsto para responder las anteriores preguntas, para hacer aplicaciones complejas basadas sobre los principios de SOA, para esto se debe elaborar los siguientes componentes en la especificación arquitectónica:

- Enterprise Services Inventory, un repositorio de programas reusables, esto con el fin de crear posteriormente los servicios.
- Enterprise Services Repository, un repositorio de servicios.
- Enterprise Services Community, un repositorio de patrones, personalizaciones de servicios y estándares.

Para lograr tener una estructura como esta:



En detalle:

User Interface (interfaz de Usuario): por lo general son muy robustas y abundantes en aplicaciones empresariales, la mayoría usando modelos, patrones o blocks ya construidos (nos referimos a componentes ya construidos y listos para usarse). En ESA definiremos los siguientes elementos:

- Centros de trabajo, que son elementos de la interfaz de usuario diseñados para conducir a un usuario por los pasos necesitados para completar un proceso.

- Centros de control, usados para gestionar los accesos a los centros de trabajo, además de usarlos para la configuración genérica de las interfaces

Process orchestration (Procesos de orquestación): la definiremos como la "lógica de proceso" que es distinta de todas las otras clases de lógica. El proceso de orquestación ocurrirá en muchos niveles diferentes. Las aplicaciones complejas la usarán como el coordinador e integrador de un set de pasos del proceso disponibles a través de los servicios de la empresa. Los proveedores de servicios como las aplicaciones existentes de la empresa destinarán flujos de trabajo para procesos dentro de su entorno y limites, y usarán estratos de integración de la lógica proceso para aceptar y enviar información hacia y desde otros sistemas. Los servicios individuales pueden usar la orquestación de proceso para crear servicios complejos. La meta en ESA es simplificar la orquestación de los procesos a fin de que las aplicaciones pueden variarse rápidamente, a un bajo precio

Existen dos formas de orquestación de procesos:

- La orquestación de proceso Frontend (interfaz de usuario) que es conversacional. Está enfocada a la colaboración y la interacción con usuarios. Una tecnología llamada *"procesos guiados"* que pueden guiar a un usuario a través de un set de pasos que pueden requerir muchas aplicaciones diferentes de la empresa. Los métodos dirigidos son diseñados para permitirle mantener fácilmente un contexto para un proceso. Son también flexibles y le dejan cambiar el flujo de proceso por una instancia de un proceso al vuelo.
- La orquestación Backend son procesos transaccionales largos, primordialmente asincrónicos. Un ejemplo de orquestación de proceso del backend es la funcionabilidad comercial de un componente de administración por procesos en la Infraestructura de SAP NetWeaver Exchange, el cual tiene que un motor de proceso de negocio de alto rendimiento, robusto para los servicios de la empresa para automatizar procesos complicados.

Enterprise services (Servicios Empresariales)

Los servicios de la empresa vienen en muchas formas. Usualmente el termino *servicio empresarial* se refiere a los servicios que quedan expuestos por las aplicaciones empresariales o por proveedores de servicios BPM.

Aunque los servicios de la empresa se basan en los estándares de webservices no son lo mismo, no necesariamente en simplemente la exposición de la funcionalidad a cualquier nivel de granularidad. Los servicios empresariales viven en el Enterprise Services Repository que almacena:

- La descripción de las interfaces de los servicios.
- Información del uso de estos servicios dentro del dominio del modelo.
- La relación de los servicios empresariales y los procesos comerciales.

Business objects (Entidades del Negocio)

Son las colecciones relacionadas a datos y funcionalidad dentro servicio proveedor. Los objetos de negocio también pueden ser unidades de modelado

En la forma más pura de ESA, los servicios empresariales son extensiones de objetos comerciales. En otras palabras, los servicios de la empresa exponen la funcionalidad de objetos comerciales para el exterior.

Uno de los retos principales de ESA es que los servicios empresariales deben forjarse en aplicaciones de la empresa y otros proveedores de servicios que no fueron originalmente diseñados para ESA. Estas aplicaciones no tienen el equivalente de objetos comerciales dentro de ellas que facilitan construir servicios. El reto de construir servicios empresariales sobre aplicaciones empresariales basadas en la pila mainframe/server/client es que la funcionalidad de la aplicación no es separada en trozos reusables. Al crear un servicio reusable empresarial, uno debe entender que puede tener algunos efectos secundarios no deseados

Distributed persistence (Persistencia distribuida)

En ESA la centralización de información en una sola base de datos para una arquitectura de tirpo mainframe/server/client ya no es válida, se requiere un repositorio distribuido a pesar de la existencia de cierto nivel de redundancia.

Esto es posible a través de mecanismos de agregación y distribución que permiten construir virtualmente una estructura lógica y unificada sobre una colección de distintos repositorios distribuidos. Pero esto no es suficiente los mecanismos distribuidos de transacción de cada base de datos deben coordinar la actualización de todos los registros en los distintos repositorios en los cuales se instancia o se aloja

Conclusión.

A través de estos conceptos y los posibles modelos arquitectónicos que los adoptan, empezaron a aparecer nuevas soluciones para la gestión del flujo comercial de una organización, mismas que consideran herramientas de monitoreo e integración, toda esta gama de conceptos intenta responder a la siguiente interrogante:

"¿Es posible integrar sistemas de información de gran escala o empresariales, conforme al cambio de los procesos de negocio?"

Expuesto esto presentamos un nuevo concepto al cual denominaremos ESB (Enterprise Service BUS) que nos permitirá resolver temas como la unificación en el ciclo productivo a través de la integración de los servicios empresariales de una organización logrando objetivos como:

- Desacoplamiento de los servicios.

- Ampliación y mediación de políticas de los servicios empresariales.
- Producir desarrollos de soluciones centradas en las políticas de la institución.
- Estandarización de transformación de mensajes.
- Extensibilidad de los servicios en distintos puntos del flujo comercial de la institución.
- Control de tráfico y priorización de prestación de servicios.
- Reducción de costo al encarar un cambio operativo.
- Flexibilidad a los cambios de procesos operativos.

2. PLANTEAMIENTO DEL PROBLEMA

Actualmente varias instituciones cuentan con una infinidad de sistemas de información, los cuales tienen que ser integrados entre sí, por lo cual actualmente se optar una arquitectura orientada a servicios para la construcción de las mismas.

Sin embargo a medida que se adiciona nuevos aplicativos para automatizar los procesos de la cadena de producción, existen un infinidad de problemas que se presentan como:

- Dificultad en la unificación de una cadena de producción automatizada.
- Falta de cohesión en los procesos automatizados.
- Problemas de integración entre servicios construidos en diferentes tipos de tecnología
- Disonancia semántica entre los sistemas propuestos.
- Dificultad a los momentos de centralizar la información.
- Alto grado de covarianza de procesos del flujo comercial

2.1 FORMULACION DEL PROBLEMA

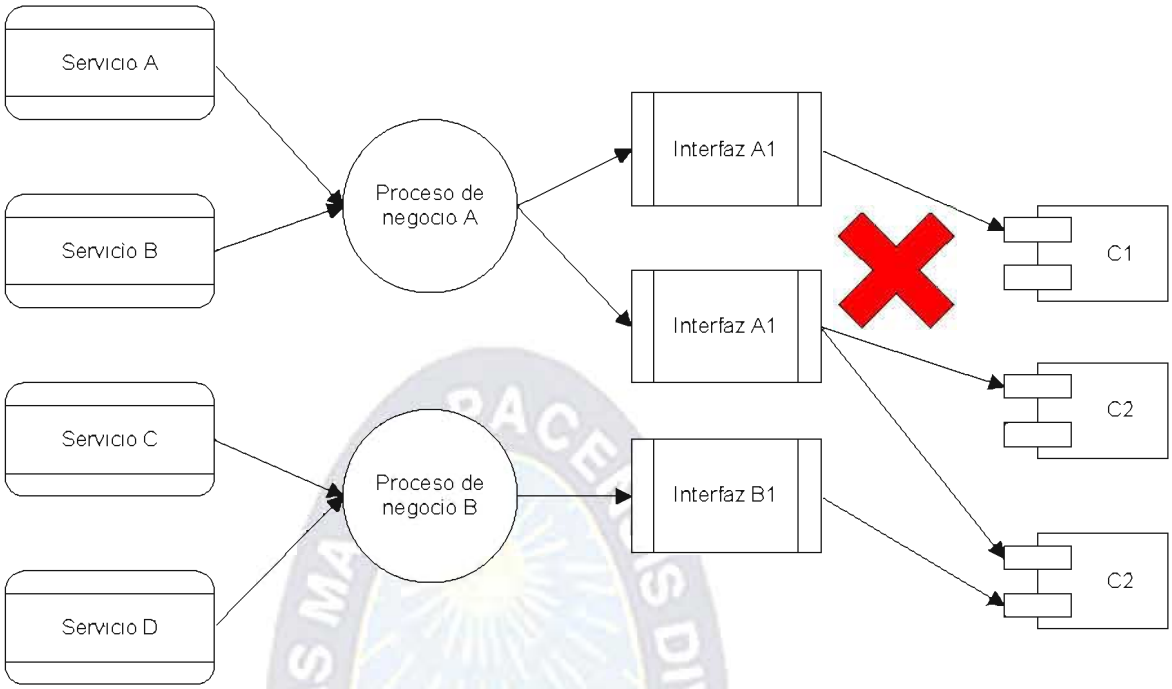
Considerando los problemas expuestos anteriormente, nos formulamos la siguiente pregunta:

"¿Es posible integrar los procesos automatizados del flujo comercial de una organización, considerando como factor primordial los cambios constantes que sufren los mismos?"

2.2 SISTEMATIZACION DEL PROBLEMA

Para poder sistematizar nuestro dominio del problema analizaremos los factores que se presentan concurrentemente durante la construcción de los aplicativos que adoptan SOA. Detallamos:

Mala exposición de los servicios, por el uso de contratos ambiguos:



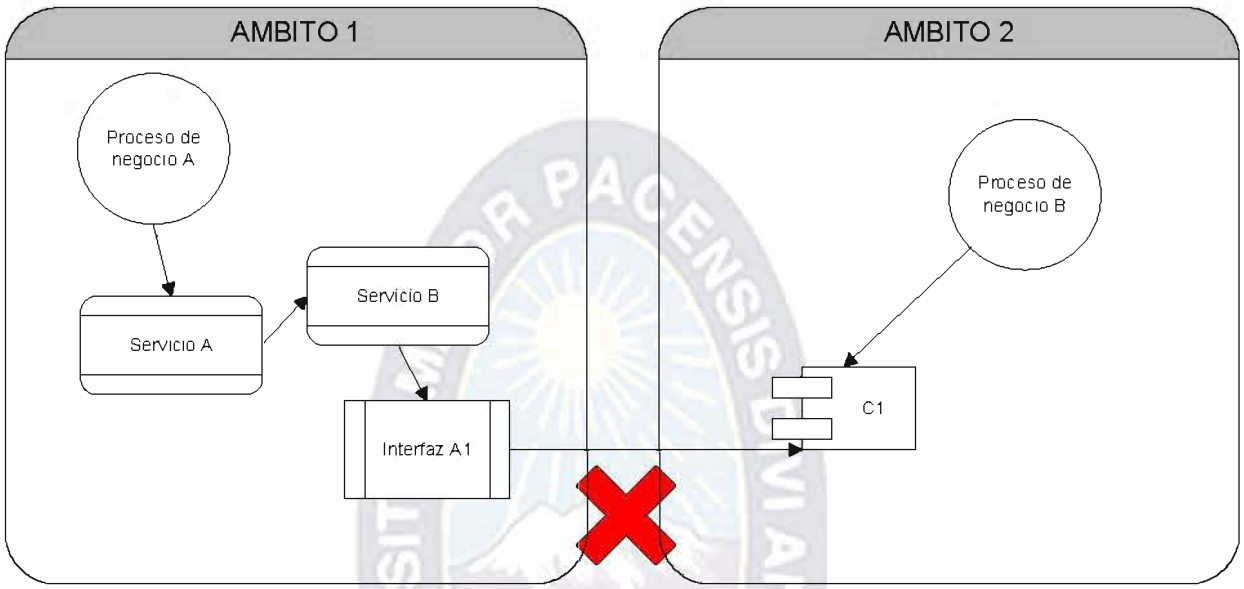
Donde se entiende que:

- Los servicios son unidades lógicas encapsulan funcionalidades peculiares
- Los procesos de negocio son componentes más complejos que los servicios que encapsulan parte de la cadena productiva de una organización.
- Las interfaces son unidades agiles que permiten exponer dichos proceso de negocio.
- Las figuras que tienen nombres como C1, C2, etc. son los contratos mediante los cuales expondremos nuestras interfaces, mimos que permitirán intercambiar información entre las interfaces y el componente que los consuma.

Observando el anterior gráfico se entiende el proceso de negocio A cuenta con más de una interfaz para exponerlo (generalmente por temas de extender la funcionalidad del servicio); esto puede causar un problema que denominaremos **ambigüedad en la semántica del servicio** que conlleva los siguientes problemas:

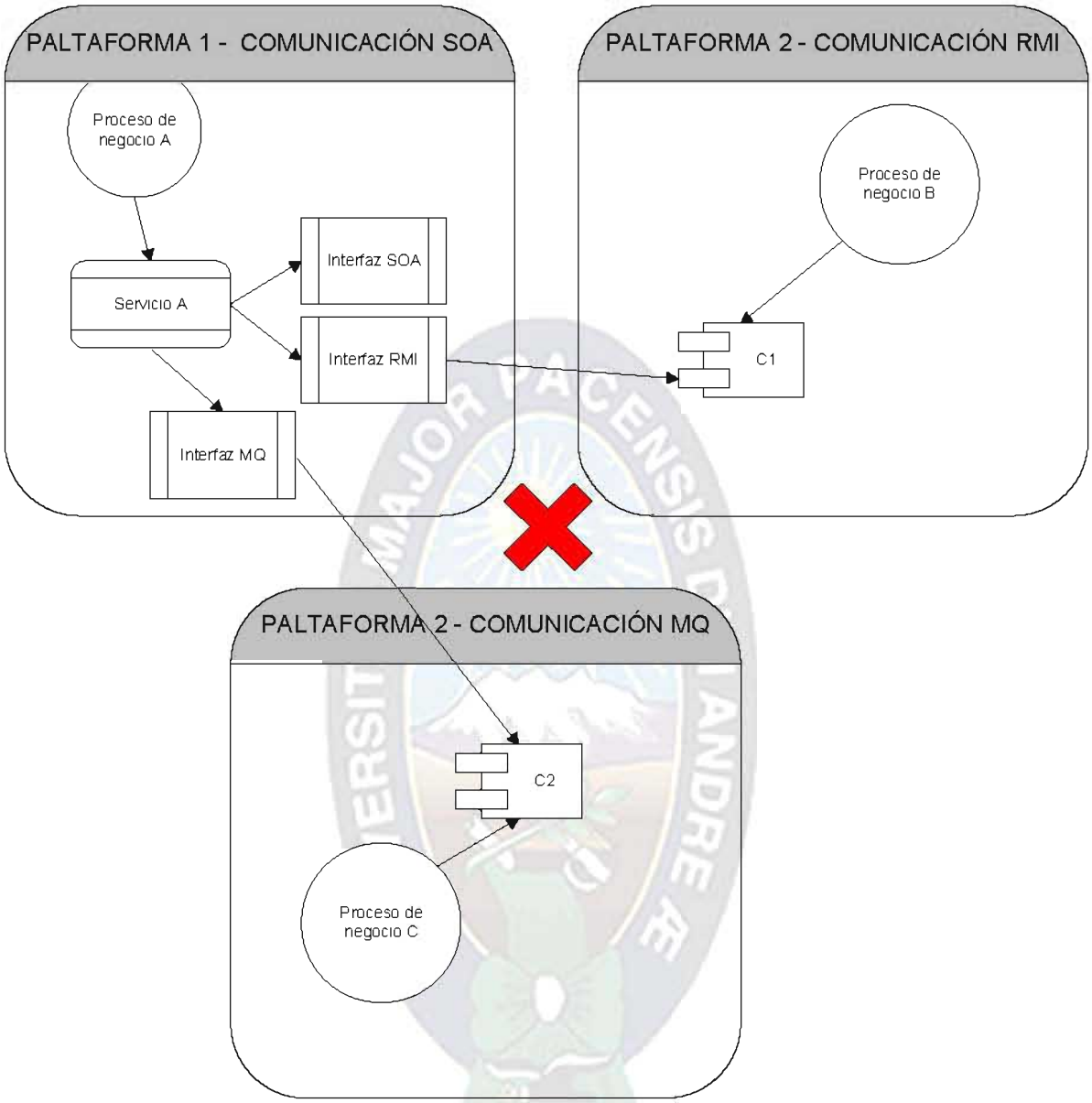
- Mala exposición de las interfaces ya que un mismo contrato da a exponer dos procesos de negocios causando ambigüedad en sí mismo.
- Cada punto de entrada o interfaz de un proceso de negocio deja de representar semánticamente la funcionalidad del proceso de negocio.
- Se pierde la escalabilidad del servicio, por el factor de crecimiento excesivo de las interfaces.

Creación de servicios intermedios que no reflejan la lógica del negocio, si no solamente cumplen el rol de la exponer de información inaccesible, lo cual baja la cohesión del sistema:



Observando el anterior gráfico se entiende que el servicio B no tiene asociado un proceso de negocio como tal (por lo cual se entiende que no llega cumplir con todas las características necesarias para ser un servicio, según las definiciones explicadas anteriormente específicamente en el punto 5 de la introducción). Este problema radica generalmente durante la orquestación del flujo comercial entre distintos ámbitos (Por ejemplo inter-instituciones o inter-aplicaciones); como vemos el componente 1 (C1) se encuentra en otro ámbito y requiere coordinar la transformación entre el proceso de negocio A y B, este problema se puede extender no solo en calidad del servicio si en el numero de servicios que tengamos necesitemos exponer.

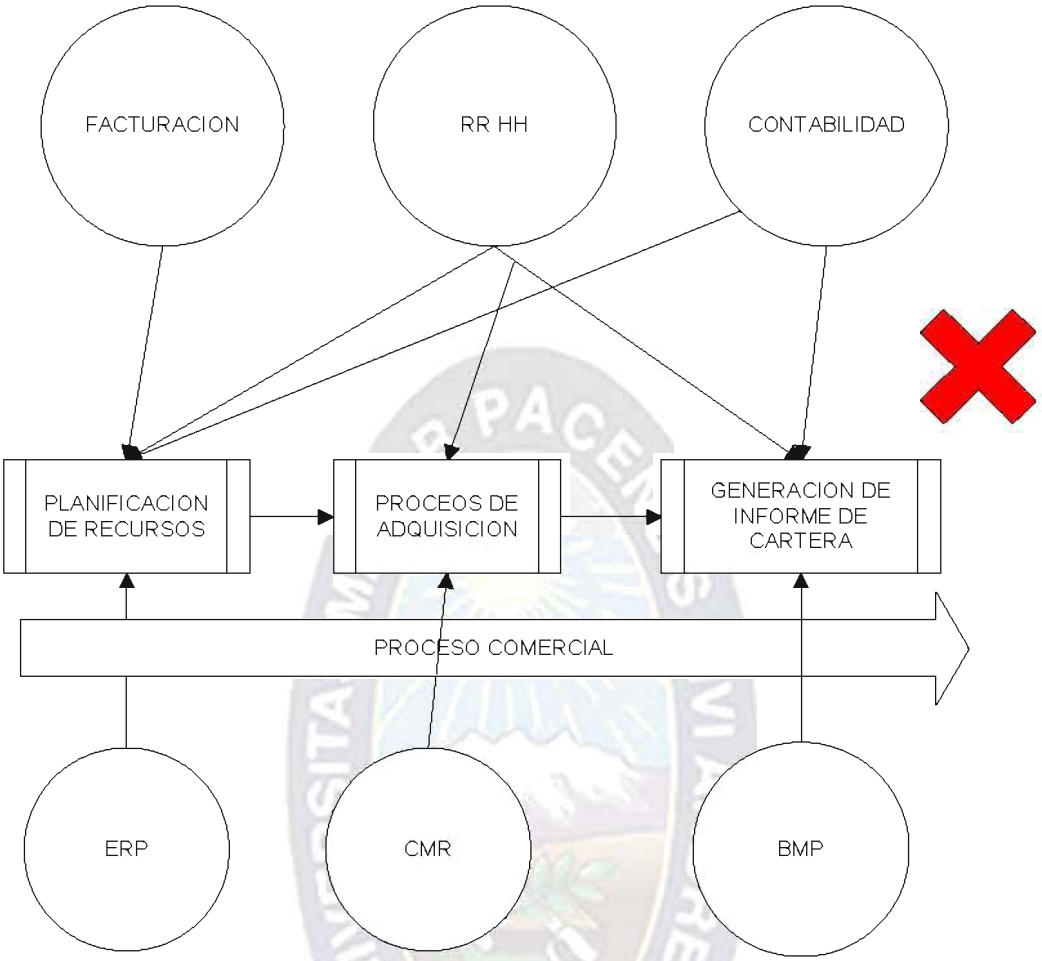
Problemas al exponer un servicio por más de un protocolo de aplicación:



Observando el grafico se entiende que se requiere desarrollar más de un adaptador, esto incrementa el tiempo de desarrollo y convierte a un esquema flexible en un entorno de comunicación complejo.

Motivos como el alto costo (económico y en tiempo) de desarrollo resultan en el no cumplimiento de objetivos en proyectos de implementación de nuevos aplicativos

Problemas en la orquestación del flujo comercial, esto se debe a la generación de distintos núcleos lógicos dependientes entre sí, lo cual puede ocasionar alto acoplamiento durante el flujo comercial del sistema:



Como se ve en el gráfico varios sistemas de información están interconectados a los procesos de negocio del flujo comercial de forma directa, causando un alto grado de acoplamiento, esto queda a evidencia en el caso de que el proceso de planificación de recursos requiera cambiar de orden, llegando a estar posterior al proceso de adquisición causando un efecto colateral en todos los sistemas de información del cual dependa ya que en varios casos las unidades lógicas esta sincronizadas.

La falta de flexibilidad dentro del flujo comercial de una organización, por lo general la hace menos competente en el mercado.

2.3 PRONOSTICO

Una institución que presente los problemas mencionados tendrá que asumir factores como:

1. Exceso de servicios y/o interfaces de integración, lo cual es causante de un costo alto de mantenimiento del software.

2. Costo innecesario en la construcción de adaptadores para servicios que se expongan por más de un protocolo de aplicación (MQ, SOA, etc.).
3. Tener variaciones del flujo comercial, que ocasiona ambigüedades en la transformación de la información.
4. Inflexibilidad en los procesos de negocio, por lo tanto difícil adaptación a los cambios del mercado.
5. Inconvenientes en el manejo de la concurrencia de servicios que no fueron planificados para ser consumido por un gran número de clientes.
6. Dimensionamiento erróneo de la provisión de infraestructura que soporte todos los servicios.

Estos hechos son los que se viven en varias empresas actualmente, a fuerza de la demanda creciente de productos SOA que tiende a ser mayor cada día, así lo demuestra las empresas como Northel, IBM, Microsoft, SAP y otros que proporcionan al mercado herramientas como BMP's, ERP's, AgileBepel y otros que al momento de ser implementadas tienen problemas en la integración con otros sistemas propios de la empresas adquirentes.

2.4 CONCLUSION

Analizando toda la información anterior se encuentra que las variables comunes asociados a los problemas expuestos son:

1. Gestión de Transacciones entre los servicios: Punto de control sobre la comunicación, transformación de datos y manejo de canales entre los servicios.
2. Orquestación del flujo comercial: Define los puntos neurálgicos del flujo comercial y crea unidades lógicas y colaborativas que resuelvan los mismos.
3. Abstracción de componentes lógicos de comunicación y transformación de datos entre procesos del flujo comercial

3. OBJETIVOS DE LA INVESTIGACION

Dentro de la presente investigación se identifico dentro los ámbitos empresariales y tecnológicos, factores del dominio del problema en común, que convergen en temas de diseño de procesos de negocio, gestión de proyectos de desarrollo, estandarización del flujo comercial e integración de servicios empresariales.

Es evidente que para poder resolver estos problemas se requiere tanto de un *modelo arquitectónico* como de *herramientas de integración* que permitan mitigar todos los puntos presentados en la sistematización de problema.

3.1 OBJETIVO GENERAL

"Construir de las especificaciones arquitectónicas de un Bus de Servicio Empresarial, que permita la unificación de los procesos de integración entre los servicios empresariales que componen el flujo comercial de una organización"

3.2 OBJETIVOS ESPECIFICOS

- *Adquirir flexibilidad a cambios en el flujo comercial.*
- *Integrar de sistemas construidos en diferentes plataformas.*
- *Reutilización de aplicativos LOB.*
- *Lograr transparencia en el consumo y alojamiento de los Servicios, liberando al consumidor la ubicación física (hosting) del proveedor del servicio.*
- *Alcanzar la conversión de protocolo y transporte, liberando a los consumidores de la tarea de socialización y marsahlling al momento de consumir un servicio.*
- *Realizar el ruteo de Mensajes, definiendo a donde enviar el mensaje a través de parámetros obtenidos del proveedor.*
- *Alcanzar un alto nivel de seguridad y monitoreo, permitiendo centralizar las funciones de inscripción, autenticación y autorización de los Mensajes; además de proporcionar. Además de proporcionar un ambiente de administración y control de los flujos de mensajes.*
- *Estandarizar dentro del intercambio de mensajería.*

3.3 LIMITES Y ALCANCES

Dentro del alcance del presente trabajo se encuentran:

- Elaboración de modelos arquitectónicos de Aplicativos Empresariales con un flujo comercial determinado por reglas de negocio cambiantes.
- Estudio y construcción de tecnología de integración de la información.
- Identificación de patrones de diseño comunes a los aplicativos empresariales

Se excluye:

- Estudio de Aplicativos de tipo Industrial.
- Estudio de Aplicativos monolíticos que son islas.
- Estudio de modelos de gestión de proyectos.
- Optimización de procesos de negocio.
- Definición de servicios empresariales genéricos como son ERP, SCM, CRM, etc.

4. HIPOTESIS

"Mediante la implementación de un entorno de integración de servicios, se puede obtener integridad transaccional, flexibilidad, escalabilidad y mejoras en el flujo comercial de una organización, además de una gestión controlada de políticas sobre un ecosistema de servicios caóticos en una Institución".

Identificación de variables.

El siguiente cuadro nos ofrece una vista clara de las variables:

VARIABLE	NATURALEZA	INDICADOR
Entono de integración de servicios	Estructural e Independiente	Nivel de integración que ofrece a un conjunto de sistemas
Integridad Transaccional	Cualitativa dependiente	Calidad de las transacciones y de información proporcionadas por los servicios
Flexibilidad de mejoras sobre los servicios	Cualitativa dependiente	Nivel de desacoplamiento de los servicios
Gestión de reglas políticas	Cuantitativa dependiente	Porcentaje de cumplimiento de las normas impuestas en la institución.
Flujo Comercial	Cualitativa independiente	Complejidad de la totalidad de los procesos de negocio

Relación entre variables

VARIABLE X	VARIABLE Y	TIPO DE RELACION
Entono de integración de servicios	Integridad Transaccional	Coexistencia: Si X también Y
Entono de integración de servicios	Flujo comercial	Coexistencia: Si X también Y
Entono de integración de servicios	Gestión de reglas de gerencia	Coexistencia: Si X también Y

5. JUSTIFICACION DE LA INVESTIGACION

5.1 JUSTIFICACION TEORICA

El presente trabajo pretende integrar de forma clara las soluciones tecnológica y los paradigmas arquitectónicos que se presentan dentro de una institución, logrando formalizar ciertos tipos de relaciones entre lo que posteriormente denominamos *"Inteligencia Comercial"* y la construcción de los *"Servicios Empresariales"*.

En la actualidad los términos de arquitectura, servicios empresariales, aplicaciones empresariales y desarrollo de software, van emprendiendo cuerpo y generando sub profesiones de la ingeniería de software, así mismo nuevos campos de estudios, en la investigación que llevaremos a cabo veremos varios elementos de estas nuevas ramas.

5.2 JUSTIFICACION TECONOLOGICA

El avance tecnológico en cuanto a arquitectura de software se refiere es extenso, sin embargo actualmente aun se sufre de un diseño casi artesanal de los aplicativos, consecuentemente las formas de comunicabilidad entre sistemas que divergen en su comportamiento, a partir de la presente investigación se pretende formular nuevos métodos de diseño y análisis sobre este campo de estudio, mismos que pueden responder eficaz y eficientemente a problemas genéricos de integridad del flujo comercial atreves de sistemas de información presentados en una institución que contenga variedad de aplicativos empresariales.

5.3 JUSTIFICACION PRACTICA

Desde la construcción del primer ordenador hasta la construcción de los modernos sistemas de información, siempre existió el problema de la flexibilidad a los cambios de las reglas de negocio en toda empresa que contenga un área administrativa con algún grado de automatización, es en este punto es donde se cruzan los campos del estudio de la inteligencia comercial e inteligencia de la información, y por ende la gestión de sus componentes que corresponde a cada uno respectivamente; bajo esta afirmación es necesario tener herramientas que nos permitan reflejar a un componente de software como un servicio empresarial y viceversa , pretendiendo corresponder el comportamiento de uno en el otro de forma coherente, cohesiva y sin ocasionar un alto de grado de acoplamiento cuando sea necesario; el presente trabajo pretende formalizar un diseño para este propósito.

CAPITULO II

ANALISIS Y DISEÑO

II. ANALISIS Y DEÑO

MARCO TEORICO

En esta sección veremos los temas troncales que serán la base teórica para el desarrollo de la prueba de la Hipótesis.

1. ARQUITECTURA DE SOFWARE

La arquitectura de software involucra tanto el desarrollo, metodología y los modelos de software a partir de análisis de métodos genéricos de diseño. Las estructuras de soluciones complejas de software parten de la descripción del problema para llegar a un desarrollo complejo. La arquitectura de software es el conjunto de métodos y técnicas que nos ayudaran a manejar dicho desarrollo.

La arquitectura de software es naturalmente una extensión de la disciplina de Ingeniería de Software, en términos sencillos se refiere a programar en grande. La Arquitectura de software presenta a un sistema como un conjunto de componentes y conectores, dichos componentes encapsulan una funcionalidad coherente y los conectores realizan tareas de intercomunicación en entre los componentes, la descripción de estos elementos conforman un documento al cual podemos llamar "Descripción arquitectónica". La Arquitectura de Software no es totalmente diferente a las Metodologías de Análisis y Diseño de Sistemas, más bien las complementan con perspectivas a las cuales llamamos comúnmente vistas del diseño orientado a objetos. En si la arquitectura de software de software nos ayudan a resolver Arquitecturas Empresariales, atacando puntos como la arquitectura del negocio, arquitectura tecnológica, y la arquitectura de los datos.

Evolución del desarrollo de software.

De forma drástica cada década el desarrollo de software a tenidos grandes saltos cada década en cuanto a los paradigmas se refiere. Los diseños metodológicos fueron mas complejos cada vez, unos de los primeros programas almacenados en un ordenador fueron creado en Cambridge EDSAC (programa creado solo como instrucciones binarias que eran transcritos manualmente) con el fin de realizar cálculos que un humano difícilmente podía realizar.

En 1950 ingresa el termino de subrutinas, que en realidad eran trozos de programas reusables, sin embargo estos debían ser inteligibles ante el ojo humano para cumplir su cometido, es de esa forma nace el concepto de programación de lenguaje de alto nivel dichos programas eran compilados posteriormente para estar en binario (lenguaje de maquina).

Durante la decada de los 60, el Mercado para el Software creció bastante, cada hadware venia con cierto tipo de software al cual posteriormente se lo denomino Sistema Operativo, sin embargo la complejidad que se tenia para el mantenimiento y construcción de estos complejos programas hizo que llegara lo que se conoció como " la crisis del software", hecho que permitió crear los estándares para el desarrollo de software, y de esta forma llega lo que conocemos como ingeniería de software

En 1968, Edsger Dijkstra publica un trabajo de sistemas de multiprogramación llamado "THE", el primer documento de diseño de software usando capas jerárquicas, ayudando a reducir la complejidad del software. Aunque el termino de arquitectura aun no era usado, este ya se empezaba a deslumbrar.

En 1972 David Parnas publica un trabajo sobre la modularidad de los sistemas, mismos que permitía mayor flexibilidad y compresión en tiempo de desarrollo, introduciendo el término de Información oculta, actualmente un principio fundamental para el diseño de software.

En la década de los 80s, nace la programación orientada a objetos, mediante el cual los ingenieros podían modelar tanto el problema como la codificación de la solución, los lenguajes mas populares para la implementación de este paradigma fue SmallTalk y C++, para esta década los ordenadores cambiaron de terminales basadas en texto a interfaces graficas de usuarios. Se logra introducir el termino de arquitectura de software.

En la década de los 90s llega el boom del internet, se adiciona nuevos conceptos y técnicas que buscan la relación Class/Responsibilities/Collaborators (CRC). Nacen nuevos modelos de notaciones para el análisis orientados a objetos, que incluían diagramas de transición de estados y procesos, así nace UML.

Fundamentos de la ingeniería de software.

La principal tarea de los ingenieros según Pahl (Pahl, 1996), "es aplicar su conocimiento científico e ingenio para la solución técnica del problema, y optimizarla en base al conjunto de requerimientos y limitantes sea material, legal, ambiental, tecnología, económica y humanas." Podemos extender esta definición indicando que la principal tarea de un ingeniero de software es aplicar su lógica y conocimiento en programación para la solución de los problemas de negocio.

El termino de ingeniería aplicada a software, no es del todo apropiada, ya que se asume muchas ramas en un campo tan extenso como son las disciplinas y especialidad, algunas de ellas como: diseño e implementación, SQL, Java, C++, eXtensible Stylesheet Language Transformations (XSLT) e incluso campos mas especializados que pueden considerarse como una ingeniería en si mismos.

La ingeniería de software requiere una combinación de tecnología y diseño del dominio del problema, eso implica la participación de especialistas en herramientas y métodos, es aquí donde participa la arquitectura de software, considerada también como cierto tipo de ingeniería de software. Un arquitecto de software es aquel que es un especialista en el diseño arquitectónico y es entendido en variedad de tecnologías con la finalidad de integrar estos términos para encontrar una solución cohesiva a problemas complejos.

Actualmente no es común seguir esta práctica de dividir las líneas tecnológicas, contrario se realiza un desarrollo de interfaz (UI) y un desarrollo por Back End (programación por detrás), algunos autores comenta que esta separación (llamad *corte horizontal*) no es necesariamente la más efectiva cuando cada desarrollador es dueño de un conjunto de requerimientos desde la interfaz hasta el backend.

Los objetivos primordiales en el desarrollo de software es lograr obtener productos de calidad *bajos costos*, a partir de la maximización de la productividad podemos lograr un sistema de costos eficiente, referente a la calidad la misma se logra mediante el logro de objetivos y metas del negocio, es importante tener una tasa adecuada de costo/eficacia. El mayor obstáculo para resolver este problema es la complejidad del desarrollo de software, este problema puede ser resuelto aplicando una variedad de tecnologías (frameworks y métodos de desarrollo), convirtiendo el desarrollo de software en una actividad puramente de diseño.

Usando métodos y lenguajes de tecnologías actuales podemos resolver problemas con mayor grado de certeza, de esta forma podemos romper la barrera de complejidad de la construcción de grandes sistema, como son el grado de eficacia funcional flexibilidad a cambio, portabilidad, seguridad.

Elementos de la Arquitectura de Software.

Componentes, Conectores y cualidades.

Shaw and Garlan define la arquitectura de software de forma abstracta como la descripción de los componentes de un sistema, sus iteraciones, los principales patrones del diseño del sistema, y reglas de otros patrones, por lo tanto un sistema esta definido físicamente (implementación) por sus componentes, conectores y sus iteraciones.. Booch considera un diseño orientado a objetos Otros consideran la arquitectura como un conjunto de vistas globales de alto nivel, mismas que son conocidas como referencias arquitectónicas.

Descripción arquitectónica.

Esta es la especificación de cómo, cuando se construirá un sistema, y las posibles propiedades que tiene, en otras palabras es la creación de la descripción de un sistema, que incluye las especificaciones de las cualidades del diseño en términos de estructura de software, logrando la correspondencia entre los requerimientos y dichas estructuras. Para lograr este objetivo es necesario tener varios modelos intermedios En este un rol de diseño mitológico se mapean los requerimientos funcionales independientes de los modelos o estructuras de propias del software, para poder definir el dominio del problema. Los resultados de estos modelos muestran claramente elementos no funcionales y el dominio de la aplicación funcional; dicho modelo generalmente va influenciado por factores de disponibilidad tecnológica y los requerimientos.

Arquitectura de software vs Metodologías de Diseño

La arquitectura de software es relativamente una nueva metáfora en el diseño de software, y realmente acompaña al diseño del metodológico, como a las metodologías de gestión. La arquitecta del software hoy es una combinación de papeles como diseñador del analista de sistemas e ingeniero del software. Pero la arquitectura de software es más que simplemente una

re dotación de funciones: Los aspectos diferentes de arquitectura todavía pueden ser realizados por especialistas pero ahora comúnmente pueden clasificarse como la orquestación del arquitecto principal. El concepto de arquitectura de software - se quiere decir - subsume las actividades de análisis y el diseño en un armazón mayor del diseño, más coherente. Además, las demandas de aplicaciones hoy son diferentes que lo que fue incluso 10 años atrás cuando la orientación a objetos se estaba volviendo el paradigma establecido del diseño. Las aplicaciones tienen tendencia a ser mas grandes, más integradas, y se implementa usando una gran variedad de tecnologías, las organizaciones se dan cuenta de que el alto precio de desarrollo del software.

Así para identificar esta nueva profesión de Arquitecto de software, se debe reconocer que el desarrollo del software no es realmente científico sino más bien más de cerca se parece a los gremios artesanales de la Edad Media, también se debe que el desarrollo del software no es realmente una actividad homogénea relegada para una sola especialidad (la programación) sino implica muchas especialidades y tecnologías diferentes. Si bien estas tecnologías son todo software, realmente requieren métodos de diseño. Por consiguiente, reconocemos que el Arquitecto de software implica metodologías interdisciplinarias de ingeniería del software de análisis orientado a objetos para la descomposición funcional; De programación orientada a objetos para diseño de la base de datos de relaciones y dibujo técnico XML diseño, y aun usuario interactúa y diseño de usabilidad.

Los Tipos de Arquitectura

En la industria de tecnología de la información, el término que la arquitectura se usa para referir a varias cosas. De un punto de vista empresarial, hay cuatro tipos de arquitectura:

- La arquitectura del negocio
- La arquitectura de tecnología de la información
- La arquitectura de información
- La arquitectura aplicativa (el software)

Colectivamente, estas arquitecturas son llamada arquitectura empresariales. Una negocio o arquitectura del de proceso de negocio define la estrategia comercial, la autoridad, la organización, y los procesos comerciales cruciales dentro de una empresa. El campo de proceso comercial que la reingeniería (BPR) enfoca en el análisis y que diseño de procesos comerciales, no necesariamente representada en un sistema de tecnología de la información.

La arquitectura de tecnología de la información define el hardware y los componetes del software que hacen el global sistema de información de la organización.

La arquitectura comercial es de la que se trazó un mapa para la arquitectura de tecnología de la información. La arquitectura de tecnología de la información debería permitir logro de las metas comerciales usando una infraestructura del software que soporta la adquisición, desarrollo, e implementación de aplicaciones comerciales críticas para la misión de fondo. El propósito de la arquitectura de tecnología de la información es permitirle a una compañía manejar su inversión de tecnología de la información en base a sus necesidades comerciales Esto incluye hardware y una infraestructura del software incluyendo tecnologías de la base de datos y de software

personalizado. Las tecnologías nuevas de tecnología de la información permiten capacidades y procesos comerciales que de otra manera no serían posibles. Internet es un ejemplo.

2. APLICACIONES EMPRESARIALES

Arquitectura

"La arquitectura" es un término que un montón de gente intenta definir, la mayor parte de las definiciones convergen en que una es el diseño de más alto de un sistema en sus partes; otra definición indica que hay arquitecturas múltiples en un sistema. Ralph Johnson tiene una publicación verdaderamente notable sobre la arquitectura; en esta publicación él sacó el punto que la arquitectura es una cosa subjetiva, una comprensión compartida del diseño de un sistema por los desarrolladores expertos sobre un proyecto. Comúnmente este acuerdo compartido está en la forma de los componentes principales del sistema y cómo interactúan. Al fin la arquitectura se resume a las cosas importantes _ cualquier cosa que eso sea _.

Las Aplicaciones Empresariales

Un montón de gente construye programas de computadoras a lo cual denominamos desarrollo del software. Sin embargo, hay clases bien definidas de software allí afuera, cada uno del cual tienen sus retos y sus complejidades. Las aplicaciones empresariales a menudo tienen aspectos complicados a lo que dedicarse, conjuntamente con las reglas comerciales que existen todas las pruebas de razonamiento lógico. Aunque algunas técnicas y patrones tienen importancia para toda clase de software, muchos tienen importancia para sólo una rama particular, en esta sección definiremos algunos patrones de los aplicativos empresariales.

Comenzaré con ejemplos. Las aplicaciones empresariales incluyen nómina, registros pacientes, enviándose rastreando, la cadena de análisis de costos, de puntuación de crédito, de seguro, del suministro, contabilidad, el servicio al cliente, y compraventa de moneda extranjera. Las aplicaciones de la empresa *no incluyen* inyección de combustible del automóvil, procesadores de texto, controladores del elevador, controladores químicos de la planta, interruptores telefónicos, sistemas operativos, compiladores, y juegos.

Las aplicaciones empresariales usualmente requieren datos persistentes, los datos usualmente necesitan persistir por varios años para su posterior estudio., en algún momentos deberá realizar muchos cambios en los programas que lo usan. A menudo excederá en duración el hardware que originalmente creó mucho de eso, y excederá en duración sistemas operativos y compiladores. Durante ese tiempo habrá muchos cambios para la estructura de los datos para almacenar nuevos trozos de información sin disturbar los viejos pedazos. Aun si hay una revolución y la compañía instala una aplicación completamente nueva para manejar un trabajo, los datos tienen que ser emigrados para la aplicación nueva.

Hay usualmente una buena cantidad de información _ que un sistema moderado tendrá sobre 1 GB de datos organizado en decenas de millones de registros _ que es una parte principal del sistema. Los sistemas grandes usaron indexación de estructuras de archivo como el método de acceso a la memoria virtual de IBM y el método de acceso secuencial indexado. Los sistemas modernos usualmente usan bases de datos, en su mayor parte bases de datos de relaciones. El diseño y la alimentación de estas bases de datos se han convertido en una sub profesión.

Usualmente muchas personas acceden a los datos concurrentemente. Para muchos sistemas esto puede estar cuantificado en menos de cien personas, pero para sistemas basados en la Web que hablan por la Red, esta cantidad sube en órdenes de gran magnitud. Con tantas personas hay que asegurarse que todos ellos pueden acceder al sistema correctamente y que no acceden a los mismos datos al mismo tiempo.

Con datos de cierta cantidad, hay usualmente una buena cantidad de interfaz del usuario. No es inusual tener centenares de pantallas discretas. Así, los datos tienen que ser presentados montones de formas diferentes para los propósitos diferentes. Los sistemas a menudo tienen un montón de proceso por lotes, lo cual es fácil de olvidar al enfocar la atención en casos de uso que enfatizan interacción del usuario.

Las aplicaciones de la empresa raras veces viven de una isla. Usualmente necesitan integrarse con otra empresa. Los sistemas diversos se forjan en momentos diferentes con tecnologías diferentes, y aun los mecanismos de colaboración serán diferentes: Los ficheros de datos de COBOL, CORBA, sistemas de envío de mensajes. De cuando en cuando la empresa intentará integrar a sus sistemas diferentes usando una tecnología común de comunicación. Por supuesto, casi nunca termina el trabajo, hay varios diferentes planes de integración unificados en su propio entorno. Esto se complica más cuando los negocios tratan de integrarse con sus socios comerciales también.

Aun si una compañía unifica la tecnología para la integración, incurren en problemas con diferencias en el proceso comercial y la disonancia conceptual con los datos. Una división de la compañía puede pensar que un cliente es alguien con quien tiene un acuerdo actual; Otra sucursal también cuenta eso sin tomar que no se tiene un contrato renovado; otra entiende que se tiene venta del producto cuando en verdad es una venta de servicio. Eso puede sonar fácil de solucionar, pero cuando usted tiene centenares de registros de los cuales cada campo puede tener un significado sutilmente diferente, el puro tamaño del problema se pone un reto. Como consecuencia, los datos tienen que estar todo el tiempo traducidos y escrito en toda clase de diferentes formatos sintácticos y semánticos.

Entonces allí está la materia de lo que se sujeta al término la "lógica comercial". Encuentro esto un término curioso porque hay pocas cosas que son menos lógicas que la lógica comercial. Cuando usted construye un sistema operativo usted se esfuerza por conservar completamente todo lógico. Pero las reglas comerciales le son justamente dadas, y sin esfuerzo político principal no hay nada que usted puede hacer para cambiarlos. Usted tiene que ocuparse de un conjunto imponente fortuito de condiciones extrañas que a menudo se interactúan el uno al otro en las formas sorprendentes. Por supuesto, se pusieron así por una razón: Algún vendedor ha negociado para tener un cierto pago anual dos días más tarde que usual, para cumplir con el ciclo de contabilidad de su cliente y así obtener un par de millón de dólares en el negocio. Algunas de estas miles de causas especiales y excepcionales son pistas para la llógica comercial, complicando lo que hace software comercial. En esta situación que usted tiene que organizar la lógica comercial como eficazmente como usted puede, porque una sola cosa es cierta la lógica cambiará con el paso del tiempo.

Pues algunos definen el término "Aplicación Empresarial" como un sistema grande. Sin embargo, es importante recordar que no todas las aplicaciones de la empresa son grandes, si bien le pueden proveer una buena cantidad de valor a la empresa. Las muchas personas suponen que, desde que los sistemas pequeños no son grandes, no valen la pena, y hasta cierto punto hay mérito aquí. Si un sistema pequeño falla, usualmente hace menos ruido que un sistema grande. Todavía, pienso

que tal manera de pensar tiende a darle menos de lo debido el efecto acumulativo de muchos proyectos pequeños. Si usted puede hacer cosas que mejoran proyectos pequeños, entonces ese efecto acumulativo puede ser muy significativo en una empresa, en particular desde que los proyectos pequeños a menudo tienen valor desproporcionado. Ciertamente, una de las mejores cosas que hay que usted puede hacer convertir un proyecto abrumador en uno pequeño simplificando en su arquitectura y en el proceso.

Las Clases de Aplicaciones Empresariales

Cuando discutimos cómo diseñar Aplicaciones Empresariales, y qué patrones acostumbramos a usar para eso, es importante para darse cuenta de que las aplicaciones de la empresa son todas diferentes y eso es la pista de diferentes de problemas para los diferentes modos de hacer las cosas. Tengo un conjunto de campanas de alarma que detonan cuando personas dicen, "siempre hacen esto". Para mí mucho del reto (y el interés) está en el diseño, sabiendo que alternativas existen y juzgando cuales son las más convenientes. Hay un espacio grande de alternativas entre el que elegir, pero aquí escogeré tres puntos:

Considere a un B2C (el negocio para el cliente) en detalle: Las personas hacen una lectura ligera y con suerte una compra en el carrito de compras. Para tal sistema necesitamos poder manejar un volumen muy alto de usuarios, así es que nuestra solución necesita para no ser sólo razonablemente eficiente en términos de recursos usados sino que también en el dimensionamiento a fin de que usted pueda aumentar la carga sumando más hardware. El dominio lógica para tal aplicación puede ser bastante franco: La orden captando, una cierta cantidad de fijación en los precios y enviando cálculos, y notificación de embarque. Queremos que alguien tenga el acceso al sistema fácilmente, así eso implica una presentación de Web bastante genérica que puede ser usado con el mayor alcance posible para los navegadores. La fuente de datos incluye una base de datos para mantener órdenes y quizá alguna comunicación con un sistema de inventario que ayuden con información de disponibilidad y de la entrega.
Haga contraste esto con un sistema que automatiza el procesamiento de arrendamientos. En ciertos aspectos éste es un sistema mucho más simple que el B2C porque hay muchos menos usuarios _ no más de cien o poco más o menos tal vez _. Dónde está más complicado está en la lógica comercial. Calculando proyectos mensuales de ley sobre un arrendamiento, manejando los eventos de pagos atrasados, y validando datos como un arrendamiento sea anotado en libros son todas las tareas complicadas. Un dominio comercial complicado como esto es desafiante porque las reglas son tan arbitrarias. Tal sistema también tiene más complejidad en la interfaz de usuario (la interfaz de usuario). Por lo menos esto quiere decir una interfaz mucha más compleja de HTML con más pantallas bastante complicadas. A menudo estos sistemas tienen demandas de la interfaz de usuario que conducen a los usuarios a querer una presentación más sofisticada que lo que una forma de HTML le permite, así es que una interfaz del cliente más sustancioso convencional se necesita. Una interacción del usuario más complicada también conduce al comportamiento de transacción más complicado: Anotar en libros un arrendamiento puede tomar una hora o dos, durante cuál el tiempo el usuario está en una transacción lógica. También vemos un dibujo técnico complicado de la base de datos con quizá doscientas tablas y las conexiones para paquetes para la valoración

del activo y fijación de precios.
Un tercer punto de ejemplo es un sistema que monitorea gasto simple para una compañía pequeña. Tal sistema tiene pocos usuarios y la lógica simple y fácilmente pueden ser puestos al alcance a través de la compañía con una presentación de HTML. La única fuente de datos es algunas tablas en una base de datos. Como simple en su estado actual, un sistema como esto no está desprovisto de un reto. Usted tiene que construirlo muy de prisa y tiene que tener presente que puede crecer como las personas que quieran calcular sus comprobantes de reembolso, alimentarlas en el sistema de la nómina, comprender implicaciones tributarias, proveer informes para el director financiero, relacionarse con servicios de Web de reservación de la aerolínea, etcétera. Si un negocio tiene un sistema se beneficia (como todas las aplicaciones de la empresa deberían), atrasar esos beneficios cuesta dinero. Sin embargo, usted no quiere hacer decisiones ahora que le pondrán obstáculos al crecimiento futuro. Pero si usted le suma flexibilidad ahora, la complejidad sumada para el bien de la flexibilidad en verdad lo puede hacer más duro para evolucionar en el futuro y puede atrasar implementación y así puede atrasar el beneficio. Aunque tales sistemas pueden ser pequeños, la mayoría de empresas tienen un montón de ellos así es que el efecto acumulativo de una arquitectura impropia puede ser significativo.
Cada uno de estos tres ejemplos de la aplicación de la empresa tiene dificultades, y son dificultades diferentes. Como consecuencia usted no puede sacar de entre manos una sola arquitectura que servirá para los tres. Escoger una arquitectura quiere decir que usted tiene que comprender los problemas particulares de su sistema y escoger un diseño correcto basado en su comprensión. Existen muchos de los patrones son sobre elecciones y alternativas. Aún cuando usted escoge un patrón particular, usted tendrá que modificarlo para responsabilizarse por sus demandas. Usted no puede construir software de la empresa sin pensar, y todo lo que cualquier libro puede hacer es darle más información para basar sus decisiones adelante.
Si esto se aplica a los patrones, también se aplica a las herramientas. Aunque obviamente tiene sentido para escoger tan en trozos pequeños un juego de herramientas como usted puede desarrollar aplicaciones, usted también tiene que reconocer que las herramientas diferentes son más convenientes para los propósitos diferentes. Mucho cuidado con usar una herramienta que es en realidad adecuada para una clase diferente de aplicación _ puede entrabar más que la ayuda.

Performance

Muchas decisiones arquitectónicas están basadas en el performance. Pues la mayoría de asuntos de performance se deberían pasar para cuando un sistema está en plena marcha como una medida posterior de optimización, sin embargo, algunas decisiones arquitectónicas no están preparadas para esto a posterior y aún cuando es fácil de repararse, las personas involucradas en el proyecto se preocupan por estas decisiones temprano.

Preparar un aplicativo para una posterior optimización tiene varias implicaciones, hay un corolario importante para esto: *Un cambio significativo en el performance puede invalidar algunos hechos acerca de la función*. Así, si usted mejora una nueva versión de su máquina virtual, el hardware, la base de datos, o casi cualquier cosa, usted debe rehacer sus optimizaciones de función y debe asegurarse de que todavía ayuden. En muchos casos una configuración nueva puede cambiar cosas. Ciertamente, usted puede encontrar que una optimización que usted hizo en el pasado para mejorar función en verdad lastima función en el ambiente nuevo.

Otro problema de hablar de performance es el hecho que hay tantos términos que son usados en de forma inconsistente. La víctima más notable de esto es "dimensionalidad," lo cual se usa regularmente para querer decir la mitad de docena cosas diferentes. Aquí están los términos que uso.

El tiempo de respuesta es la cantidad de tiempo que toma el sistema para el trámite de una petición del exterior. Ésta puede ser una acción de la interfaz de usuario, como presionar un botón, o una llamada de API del servidor.

La sensibilidad indica la rapidez del sistema a una petición. Esto es importante en muchos sistemas porque los usuarios pueden frustrarse si un sistema tiene sensibilidad baja, aun si su tiempo de respuesta es bueno.

La latencia es el tiempo mínimo requerido para obtener cualquier forma de respuesta, aun si el trabajo para hacerse es inexistente. Si le pido un programa no hacer nada, para decirme cuando esta sin hacer nada, entonces debería tener una respuesta casi instantánea si el programa funciona con mi computadora portátil. Sin embargo, si el programa funciona con una computadora remota, puedo ponerme a algunos segundos simplemente por el tiempo tomado para la petición y para que la misma se transmita por el cable. Como un desarrollador aplicativo, usualmente puedo no hacer nada para mejorar latencia. La latencia es también la razón por la que usted debería minimizar llamadas distantes.

El rendimiento específico es las cosas que usted puede hacer en una cantidad dada de tiempo. Si usted cronometra lo copiador de un archivo, el rendimiento específico podría ser medido en bytes por segundo. Para aplicaciones de la empresa una medida típica está transacciones por segundo (tps), pero el problema es que éste depende sobre la complejidad de su transacción. Para su sistema particular usted debería escoger un conjunto común de transacciones.

En esta terminología el performance es ya sea rendimiento específico o tiempo de respuesta _ cualquiera que tenga importancia más para usted. Algunas veces puede ser difícil de hablar de performance cuando una técnica mejora el rendimiento específico pero disminuye tiempo de respuesta, así es que es mejor usar el término más preciso. La perspectiva de sensibilidad de un usuario puede ser más importante que el tiempo de respuesta, así es que la sensibilidad perfeccionadora en un costo de tiempo de respuesta o el rendimiento específico aumentará función.

La eficiencia está en función de los recursos. Un sistema que trae a 30 tps en dos CPUs es más eficiente que un sistema que trae a 40 tps en cuatro CPUs idénticos.

La capacidad de un sistema es una indicación de máximo carga o rendimiento específico efectivo. Éste podría ser un máximum absoluto o un punto en el cual la función se zambulle debajo de un umbral aceptable.

La dimensionalidad es una medida de cómo sumar recursos (usualmente el hardware), que afecta al performance. Un sistema dimensionable es uno que le deja sumar hardware y obtener una mejora conmensurativa de performance, como doblarse cuántos servidores usted tiene para duplicar su rendimiento específico. La dimensionalidad vertical, o la escalada, indica por ejemplo la adición de más poder a un solo servidor, como más memoria.

El problema aquí es que las decisiones del diseño no afectan todo estos factores de función igualmente. Digo que tenemos corrida de dos soportes lógicos en un servidor: La capacidad del pez espada es 20 tps mientras la capacidad del Camello es 40 tps. ¿Cuál tiene mejora la función? ¿Cuál es más dimensionable? No podemos contestar la pregunta de dimensionalidad de esta información, y sólo podemos decir que el Camello es más eficiente en un solo servidor. Si

sumamos otro servidor, echamos de ver que el pez espada ahora manipula a 35 tps y el camello manipula a 50 tps. La capacidad del camello es todavía mejor. Si continuamos sumando servidores que descubriremos que el Pez Espada obtiene 15 tps por servidor adicional y que Camello obtienen 10. Dado estos datos que podemos decir que el Pez Espada tiene mejor dimensionalidad horizontal, si bien el Camello se eficiente más para menos de cinco servidores.

Al construir los sistemas de la empresa, a menudo tiene sentido construir para la dimensionalidad del hardware en vez de la capacidad o aun la eficiencia. La dimensionalidad le da la opción de mejor función si usted la necesita. La dimensionalidad también puede ser más fácil de hacer. A menudo los diseñadores hacen cosas complicadas que mejoran la capacidad en una plataforma particular del hardware cuando en verdad podría ser más barata para comprar más hardware. Si el Camello tiene un mayor costo que Pez Espada, y ese mayor costo equivale a un par de servidores, en ese entonces el Pez Espada acaba más barato aun si usted sólo necesita a 40 tps. Está de moda quejarse de tener que confiar en mejor hardware para poner en funcionamiento nuestro software correctamente, y me le uno a este coro cada vez que tengo que mejorar mi computadora portátil simplemente para maniobrar la última versión de Word. Pero el hardware más nuevo es a menudo más barato que hacer correr software en sistemas antiguos. De modo semejante, sumar más servidores es a menudo más barato antes que agregar a más programadores _ con tal que un sistema sea dimensionable.

3. PATRONES DE DISEÑO EMPRESARIALES

No hay definición general y aceptada de un patrón, pero quizá el mejor lugar para iniciar es Christopher Alexander, una inspiración para muchos entusiastas del patrón: "Cada patrón describe un problema que ocurre repetidas veces en nuestro ambiente, y entonces describe el corazón de la solución para ese problema, de tal manera que usted puede usar esta solución sobre la que un millón de veces, sin hacerlo dos veces de la misma forma". Alexander es un arquitecto, así es que él hablaba de edificios, pero la definición surte efecto bastante amable para software también. El foco del patrón es una solución particular, es que puede tratar con uno o más problemas recurrentes. Otra forma de considerarlo es que un patrón es un trozo de consejo y el arte de crear patrones es dividir muchos consejos en trozos relativamente independientes a fin de que usted pueda referirse a ellos y los pueda discutir más o menos separadamente.
Una parte crucial de patrones es que están arraigadas en la práctica. Usted encuentra patrones en lo que las personas hacen, observando las cosas en que trabajan buscando "el corazón de la solución". No es un proceso fácil, pero una vez que usted ha encontrado que algunos buenos patrones se convierten en una cosa valiosa.
Una vez que usted necesita el patrón, usted tiene que sacar en claro cómo aplicarlo a sus circunstancias. Algo crucial acerca de patrones es que usted no los puede aplicar a la solución ciegamente, motivo por el cual los patrones han sido fracasos miserables. Me gusta decir que los patrones se hornean "medio," quiero decir que usted siempre tiene que terminarlos en el horno de su proyecto. Cada vez que uso un patrón le ajusto un poco aquí y poco un allí. Usted ve la misma solución muchas veces encima, pero nunca es exactamente igual.
Cada patrón es relativamente independiente, pero los patrones no son esporádicos el uno del otro.

Si usted es un diseñador experimentado de aplicaciones de la empresa, usted probablemente se encontrará con que la mayor parte de estos patrones le son familiares.

Los patrones no son ideas originales; Son muchísima observaciones de lo que ocurre en el campo. Como consecuencia, modelamos que los autores no dicen que "inventamos" un patrón pero más bien que "descubrimos" uno. Nuestro papel es notar la solución común, buscar el tema del patrón fondo, y entonces de construir un patrón resultante adecuada a nuestra necesidad. Para un diseñador experimentado, el valor del patrón no es que le da una idea nueva; sino de ayudarle a comunicar su idea. Si usted y sus colegas saben lo que es una Remote Facade(388), usted puede comunicar bastante diciendo, " Esta clase es una Remote Facade. También le deja decir para alguien menos experto, Destinar un Objeto de Traslado de Datos". El resultado es que los patrones crean un vocabulario acerca de diseño, por lo que el nombramiento es un asunto tan importante.

La Estructura de los Patrones

Cada autor tiene que escoger su forma del patrón. Algunos basan sus clases en patrones clásicos, como de: Gang of Four, otros los fabrican.

Algo primordial es el ***nombre del patrón***. Los nombres de los patrones son cruciales, porque parte del propósito de patrones es crear un vocabulario que deja a los diseñadores comunicarse más eficazmente. Así, si le digo mi servidor de Web es basado en Form Controller y Transform View y usted sabe estos patrones, usted tiene una idea muy clara de arquitectura de mi servidor web.

Dos elementos primordiales son: la ***intensión y el bosquejo***. La intensión resume el patrón en una frase o dos; El bosquejo es una representación visual del patrón, a menudo pero no siempre un diagrama UML. La idea es crear un recordatorio conciso del patrón, así usted lo puede recordar rápidamente. Si usted ya "tiene el patrón" es decir que usted sabe la solución, sin siquiera saber el nombre, ese es entonces el intento y el bosquejo deberían ser todo lo que usted necesita para saber lo que el patrón es.

Las Limitaciones de Estos Patrones

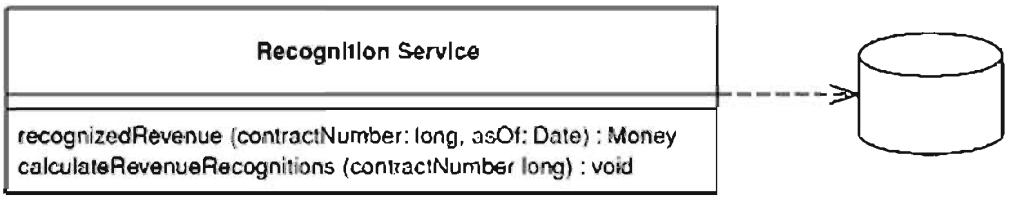
Una colección de patrones no es de ninguna manera un guía para el desarrollo de una aplicación Empresarial, los patrones es todo lo que uno que he ve en el campo laboral y son un punto de partida, no un destino final.

PATRONES DE LOGICA DE DOMINIO

PATRON TRANSACTION SCRIPT

Objetivo:

Organiza la lógica comercial en métodos donde cada procedimiento maneja una sola petición de la presentación



La mayoría de aplicaciones comerciales pueden ser consideradas como una serie de transacciones. Una transacción puede ser una vista de alguna información, en una forma particular, otra transacción será la que hará cambios para ella. Cada interacción entre un cliente y servidor contiene una cierta cantidad de lógica. En algunos casos esto puede ser tan simple como exhibir información de la base de datos. En otros puede implicar muchos pasos de validaciones y cálculos.

Un Transaction Script organiza toda esta lógica primordialmente como un solo método, haciendo llamadas directamente para la base de datos o a través de una envoltura delgada de la base de datos. Cada transacción tendrá su "Transaction Script", aunque las sub tareas comunes pueden ser cortadas en sub métodos.

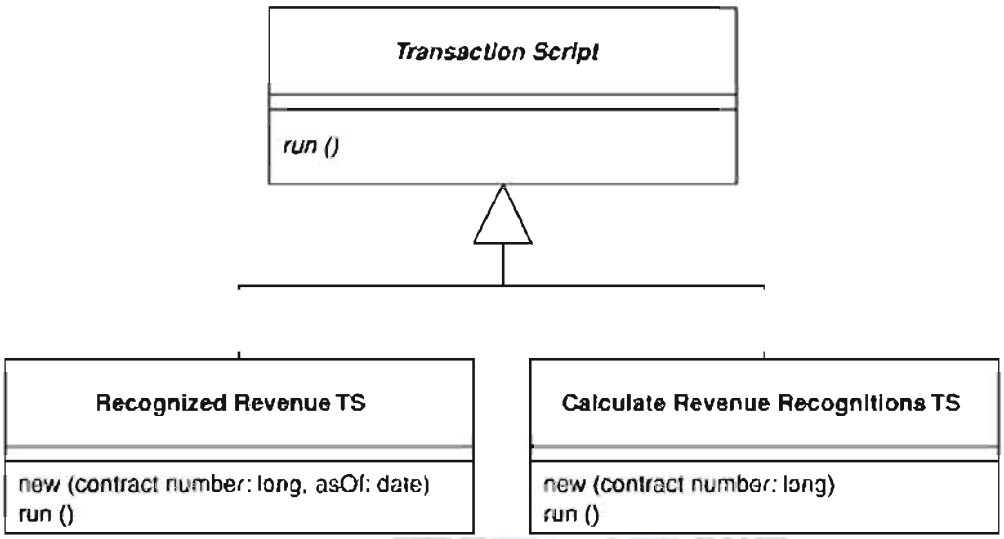
Como trabaja:

Primordialmente organiza las transacciones que se realiza en un sistema. Si su necesidad es reservar un cuarto de hotel, la lógica para comprobar la disponibilidad del cuarto, calcular tasas, y actualizar la base de datos es encontrada adentro el método del Cuarto de Hotel del Libro.

Para los casos simples no hay mucho qué decir cómo organiza usted esto. Por supuesto, al igual que con algún otro programa usted debería estructurar el código en módulos.

Uno de los beneficios de este acercamiento es que usted no necesita preocuparse por lo que otras transacciones marchan. Su tarea es tener el aporte, interrogar la base de datos y salvar sus resultados para la base de datos.

Usted puede organizar sus Trasaction Script en las clases de dos formas. Lo más común debe tener varios Trasaction Script en una sola clase, donde cada clase define un área sujeto a Script relacionados a un Transacción. La otra forma es tener cada script de Transacción en su propia clase, utilizando el patrón de Command Gang of Four . En este caso usted define una superclase para sus comandos y se especifica la forma de ejecución y la cantidad de veces que se lo hace evitando ataques lógicos al Script Transaction. La ventaja de esto es que le deja manipular instancias de script como objetos en runtime.



Cuando para Usarlo

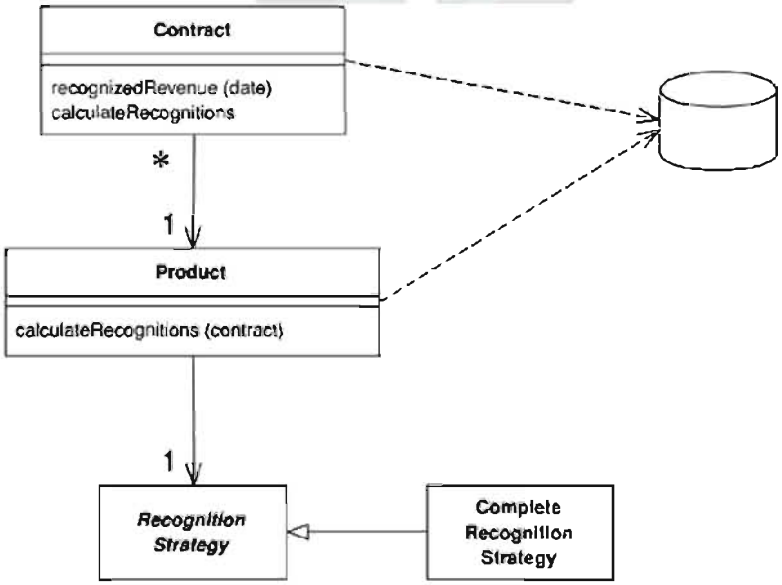
La ventaja de un Script de Transacción es su simplicidad. Organizar lógica así es natural para aplicaciones con sólo un poco de lógica, con un costo mínimo de desarrollo.

Como la lógica comercial se pone más complicada, sin embargo se consigue progresivamente dándole forma en el lugar adecuado del diseño. Un problema particular a esperar es la duplicación entre transacciones.

PATRON DOMINIO DEL MODELO

Objetivo:

Incorporar la lógica comercial en un objeto de modelo del dominio tanto en su comportamiento como en los datos.



Con una mala lógica comercial esto puede ser muy complejo. Las reglas y la lógica describen muchas inclinaciones y casos diferentes de comportamiento, y es esta complejidad define si los objetos diseñado funcionaran o no. Un Modelo de Dominio crea una trama de objetos interconectados, donde cada objeto representa a algún concepto significativo, ya sea tan grande como una corporación o tan pequeño como una línea de código.

Cómo Trabaja

Implementar un Modelo de Dominio en una aplicación implica intercalar un estrato entero de objetos que modelan el área comercial. Usted encontrará que los objetos que imitan los datos en el negocio y objetos que captan las reglas los usos comerciales. En su mayor parte los datos y proceso están combinados para aglomerar los procesos cerca de los datos con los que ellos trabajan.

Un modelo de dominio OO a menudo se parecerá mucho a un modelo de la base de datos, pero todavía tendrá un montón de diferencias. Un Modelo de Dominio entremezcla datos y proceso, tiene atributos multipreciados y una trama complicada de asociaciones, y usa herencia.

Como consecuencia veo dos estilos de Modelo de Dominio en el campo. Un Modelo simple de Dominio se parece mucho al diseño de la base de datos con en su mayor parte un objeto de dominio para cada mesa de la base de datos. Un Dominio enriquecedor Model puede mirar desemejante del diseño de la base de datos, con herencia, las estrategias, y otros patrones Gang of Four, y tramas complicadas de en trozos pequeños objetos interconectados. Un Modelo enriquecedor de Dominio es mejor para la lógica más complicada, pero es más duro hacer mapas para la base de datos. Un Modelo simple de Dominio puede usar Registro Activo (160), mientras que un Modelo enriquecedor de Dominio requiere a Data Mapper (165)

Desde que el comportamiento del negocio está sujeto a una buena cantidad de cambio, es importante poder modificar, construir, y probar este estrato fácilmente. Como consecuencia usted querrá el mínimo de parearse del Modelo de Dominio para otros estratos en el sistema. Usted echará de ver que una fuerza orientadora de muchos patrones de la acodadura es mantener como pocas dependencias tan posible entre el modelo de dominio y otras partes del sistema.

Con un Modelo de Dominio está un gran número de alcances diferentes que usted podría usar. El caso más simple es solo una aplicación de usuario donde toda la gráfica del objeto es leída de un archivo y puesta en la memoria. Metiendo cada objeto en la memoria consume demasiada memoria y tarda demasiado tiempo. La belleza de bases de datos orientadas a objetos es que le dan la impresión de estar moviendo objetos entre la memoria y el disco.

Sin una base de datos OO usted tiene que hacer esto usted mismo. Usualmente una sesión implicará tirar hacia adentro una gráfica del objeto de todos los objetos involucrados en ella. Éste ciertamente no serán todos los propósitos y usualmente no todas las clases. Así, si usted tiene a la vista un set de contratos que usted podría jalar en sólo los productos para los que se estableció referencias de por ahí se contrae dentro de su juego operativo. Si usted está simplemente cálculos que realizan en contratos y el reconocimiento de ingresos objeta,

usted no puede tirar hacia adentro cualquier objetos del producto del todo. Exactamente lo que usted tira en la memoria es gobernado por su base de datos trazando un mapa de objetos.

Si usted necesita la misma gráfica del objeto entre las llamadas para el servidor, usted tenga que ahorrar el estado del servidor a alguna parte, lo cual es el tema de la sección adelante salvo estado del servidor (página 81).

El problema con comportamiento específico en uso separante es que puede conducir a la duplicación. El comportamiento que es separado de la orden es más duro para encontrar, así es que las personas no tienden a no sede ella y la duplican en lugar de eso. La duplicación rápidamente puede conducir a más complejidad y más inconsistencia, pero me he encontrado con que la sensación de plenitud acontece mucho menos a menudo que previsto. Si eso ocurre, es relativamente fácil de ver y no difícil para repararse. Mi consejo es no separar comportamiento específico en uso. Meta todo ello en el objeto que es el ataque natural. Fije la sensación de plenitud cuándo, y si, se convierte en un problema.

MARCO REFERENCIAL

4. DISEÑO DE SERVIVIOS DISTRIBUIDOS

Objetivos del diseño de aplicaciones distribuidas

El diseño de una aplicación distribuida implica la toma de decisiones sobre su arquitectura lógica y física, así como sobre la tecnología e infraestructura que se emplearán para implementar su funcionalidad. Para tomar estas decisiones, debe tener un conocimiento claro de los procesos empresariales que realizará la aplicación (sus requisitos funcionales), así como los niveles de escalabilidad, disponibilidad, seguridad y mantenimiento necesarios (sus requisitos no funcionales, funcionales u operativos).

El objetivo consiste en diseñar una aplicación que:

- Solucione el problema empresarial para el que se diseña.
- Tenga en consideración la seguridad desde el principio, teniendo en cuenta los mecanismos adecuados de autenticación, la lógica de autorización y la comunicación segura.
- Proporcione un alto rendimiento y esté optimizada para operaciones frecuentes entre patrones de implementación.
- Esté disponible y sea resistente, capaz de implementarse en centros de datos de alta disponibilidad y redundantes.
- Permita la escalabilidad para cumplir las expectativas de la demanda y admita un gran número de actividades y usuarios con el mínimo uso de recursos.
- Se pueda administrar, permitiendo a los operadores implementar, supervisar y resolver los problemas de la aplicación en función del escenario.
- Se pueda mantener. Cada parte de funcionalidad debería tener una ubicación y diseño predecibles teniendo en cuenta distintos tamaños de aplicaciones, equipos con conjuntos de habilidades variadas y requisitos técnicos y cambios empresariales.
- Funcione en los distintos escenarios de aplicaciones y patrones de implementación.
- Las instrucciones de diseño que se ofrecen en los siguientes capítulos persiguen estos objetivos y explican los motivos para las decisiones de un diseño en particular siempre que sea importante para entender su fondo.

Servicios e integración de servicios

A medida que crece Internet y las tecnologías relacionadas, y las organizaciones buscan integrar sus sistemas entre límites de departamentos y de organización, ha evolucionado un enfoque de generación de soluciones basado en servicios. Desde el punto de vista del consumidor, los servicios son conceptualmente similares a los componentes tradicionales, salvo que los servicios encapsulan sus propios datos y no forman parte, estrictamente hablando, de la aplicación sino que

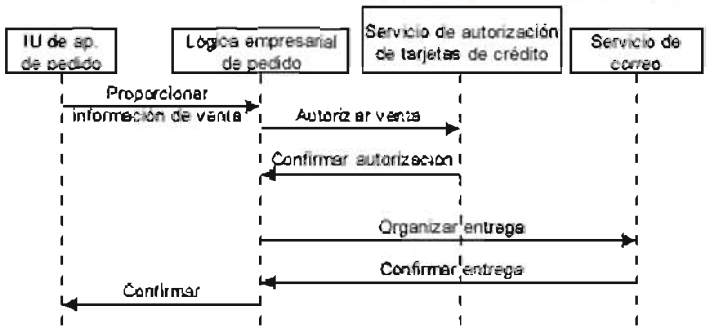
son utilizados por ésta. Aplicaciones y servicios que necesitan integrarse se pueden generar en distintas plataformas, por distintos equipos, en diferentes programas y se pueden mantener y actualizar de forma independiente. Por tanto, es esencial que implemente la comunicación entre ellos con el mínimo acoplamiento.

Se recomienda que implemente la comunicación entre los servicios empleando técnicas basadas en mensajes para proporcionar altos niveles de solidez y escalabilidad. Puede implementar la comunicación de mensajes de forma explícita (por ejemplo, escribiendo código para enviar y recibir mensajes de Message Queue Server), o bien, puede utilizar componentes de infraestructuras que administran la comunicación de forma implícita (por ejemplo, con un servidor proxy de servicios Web generado por

Microsoft Visual Studio® .NET).

Los servicios exponen una interfaz de servicios a la que se envían todos los mensajes entrantes. La definición del conjunto de mensajes que se deben intercambiar con un servicio para que éste realice una tarea empresarial específica constituye un contrato. Puede imaginarse una interfaz de servicios como una fachada que expone la lógica empresarial implementada en el servicio para consumidores potenciales.

Por ejemplo, considere una aplicación comercial de venta a través de la cual los clientes solicitan productos. La aplicación utiliza un servicio de autorización de tarjetas de crédito externas para validar los detalles de la tarjeta de crédito del cliente y autorizar la venta. Una vez comprobados los datos de la tarjeta de crédito, se utiliza un servicio de correo para organizar la entrega de los productos. El siguiente diagrama de secuencias muestra este escenario.



En este escenario, el servicio de autorización de las tarjetas de crédito y el servicio de correo desempeñan cada uno una función en el proceso empresarial global de compra. A diferencia de los componentes ordinarios, los servicios existen en sus propios límites de confianza y administran sus propios datos, fuera de la aplicación. Por tanto, debe estar seguro de establecer una conexión segura y autenticada entre la aplicación de llamada y el servicio cuando utilice un enfoque basado en servicios para el desarrollo de aplicaciones. Además, podría implementar la comunicación mediante el uso de un enfoque basado en mensajes, haciendo el diseño más adecuado para describir procesos empresariales (a veces denominados transacciones empresariales o

transacciones de ejecución larga) y para el acoplamiento flexible de sistemas que son frecuentes en soluciones distribuidas de gran tamaño, especialmente si el proceso empresarial implica varias organizaciones y distintas plataformas.

Por ejemplo, si las comunicaciones basadas en mensajes se utilizan en el proceso mostrado, el usuario puede recibir la confirmación del pedido segundos u horas después de que se proporciona la información de venta, dependiendo de la capacidad de respuesta de los servicios de autorización y entrega. La comunicación basada en mensajes permite también realizar el diseño de la lógica empresarial de forma independiente al protocolo de transporte subyacente utilizado entre los servicios.

Si la aplicación utiliza un servicio externo, la implementación interna del servicio le es indiferente al diseño; siempre que el servicio realice lo que se supone que debe realizar. Simplemente necesita saber la funcionalidad empresarial que ofrece el servicio y los detalles del contrato que debe respetar para comunicarse con el mismo (como el formato de comunicación, esquema de datos, mecanismo de autenticación, etc.). En el ejemplo de la aplicación comercial, el servicio de autorización de tarjetas de crédito ofrece una interfaz a través de la cual se pueden pasar al servicio los detalles sobre la venta y la tarjeta de crédito, así como la respuesta indicando si se aprueba o no la venta. Desde la perspectiva del diseñador de la aplicación comercial, lo que sucede dentro del servicio de autorización de tarjetas de crédito es irrelevante; lo único que importa es determinar qué datos es necesario que se envíen al servicio qué respuestas se recibirán del servicio y cómo comunicarse con el servicio.

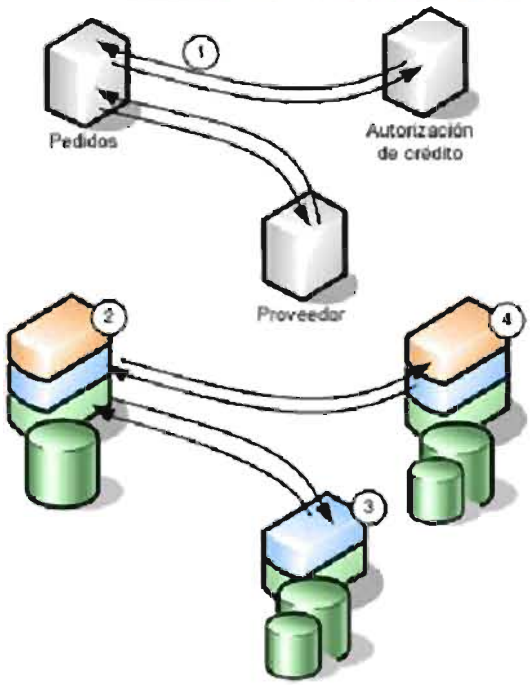
Internamente, los servicios contienen varios tipos de componentes comunes a las aplicaciones tradicionales. (El resto de esta guía se centra en los distintos componentes y su función en el diseño de la aplicación.) Los servicios contienen componentes de lógica que organizan las tareas empresariales que realizan, los componentes empresariales que implementan la lógica empresarial real del servicio y los componentes de acceso a datos que tienen acceso al almacén de datos del servicio. Además, los servicios exponen sus funcionalidades a través de interfaces de servicio, que controlan la semántica utilizada para exponer la lógica empresarial subyacente. La aplicación también llamará a otros servicios a través de los agentes de servicios, que se comunican con el servicio de parte de la aplicación cliente que realiza la llamada.

Aunque los servicios basados en mensajes se pueden diseñar para que se llamen sincrónicamente, puede resultar ventajoso generar interfaces de servicios asincrónicos, que permiten un enfoque de acoplamiento flexible en el desarrollo de aplicaciones distribuidas. El acoplamiento flexible que ofrece la comunicación asincrónica posibilita la generación de soluciones de alta disponibilidad, escalabilidad y duración formada por servicios existentes. Sin embargo, un diseño asincrónico no proporciona estas ventajas de forma gratuita: el uso de la comunicación asincrónica indica que el diseño puede necesitar tener en cuenta consideraciones especiales como la correlación de mensajes, la administración de concurrencia de datos optimista, la compensación de procesos empresariales y la no disponibilidad de servicios externos.

Componentes y niveles en aplicaciones y servicios

Se ha convertido en un principio ampliamente aceptado en el diseño de aplicaciones distribuidas la división de la aplicación en componentes que ofrezcan servicios de presentación, empresariales y de datos. Los componentes que realizan tipos de funciones similares se pueden agrupar en capas, que en muchos casos están organizados en forma de apilamiento para que los componentes que se encuentran por "encima" de una capa determinada utilicen los servicios proporcionados por ésta, y un componente especifico utilizará la funcionalidad proporcionada por otros componentes de su propia capa, y otras capas "inferiores", para realizar su trabajo.

Esta visión dividida de una aplicación también se puede aplicar a los servicios. Desde un punto de vista de alto nivel, se puede considerar que la solución basada en servicios está formada por varios servicios, los cuales se comunican entre sí pasando mensajes. Desde el punto de vista conceptual, los servicios se pueden considerar como componentes de la solución global. Sin embargo, internamente el servicio está formado por componentes de software, al igual que cualquier otra aplicación, los cuales se pueden agrupar de forma lógica en servicios de presentación, empresariales y de datos, tal y como se muestra en la figura:



Los aspectos importantes que se deben tener en cuenta de esta figura son los siguientes:

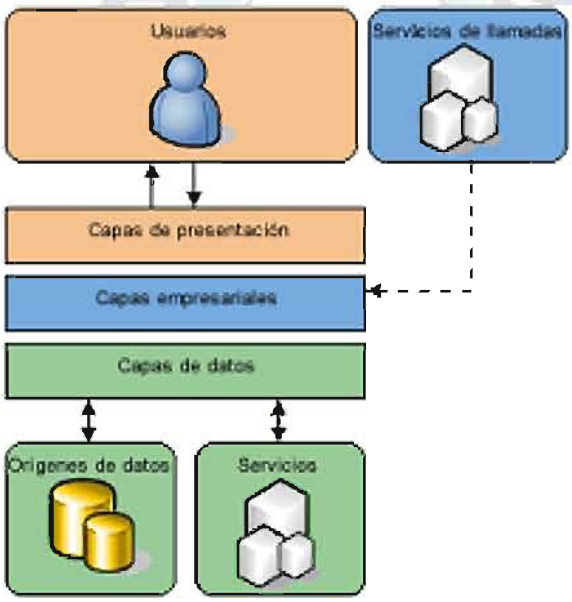
1. Los servicios se diseñan generalmente para comunicarse entre sí con el mínimo grado de acoplamiento. El uso de la comunicación basada en mensajes ayuda a desacoplar la disponibilidad y escalabilidad de los servicios, y basarse en los estándares de la industria, como los servicios Web XML, permite la integración con las demás plataformas y tecnologías.

2. Cada servicio está formado por una aplicación que dispone de sus propios orígenes de datos, lógica empresarial e interfaces de usuario. Un servicio puede presentar el mismo diseño interno que una aplicación tradicional de tres niveles.

3. Puede generar y exponer un servicio que no disponga de una interfaz de usuario directamente asociada (un servicio diseñado para que lo invoquen otras aplicaciones a través de una interfaz de programación).

4. Cada servicio encapsula sus propios datos y administra las transacciones atómicas con sus propios orígenes de datos.

Es importante tener en cuenta que las capas son simplemente agrupaciones lógicas de los componentes de software que conforman la aplicación o servicio. Ayudan a diferenciar entre los distintos tipos de tareas que realizan los componentes, facilitando el diseño de la reutilización en la solución. Cada capa lógica contiene un número de tipos de componentes discretos agrupados en subcapas, cada una de las cuales realiza el mismo tipo de tarea específica. Al identificar los tipos genéricos de componentes que existen en la mayoría de las soluciones, puede construir un mapa coherente de una aplicación o servicio y, a continuación, utilizar este mapa como plano técnico para el diseño.



Una solución distribuida puede que necesite abarcar varias organizaciones o niveles físicos, en cuyo caso tendrá sus propias directivas en relación a la seguridad, administración operativa y comunicaciones de la aplicación. Estas unidades de confianza, o zonas, pueden ser un nivel físico, un centro de datos o un departamento, sección o empresa que tenga estas directivas definidas. Unidas, estas directivas definen reglas para el entorno en el que se implementa la aplicación y la forma en que los niveles del servicio o aplicación se comunican. Las directivas abarcan toda la aplicación y la forma en que se implementan afecta a las decisiones sobre el diseño en cada nivel.

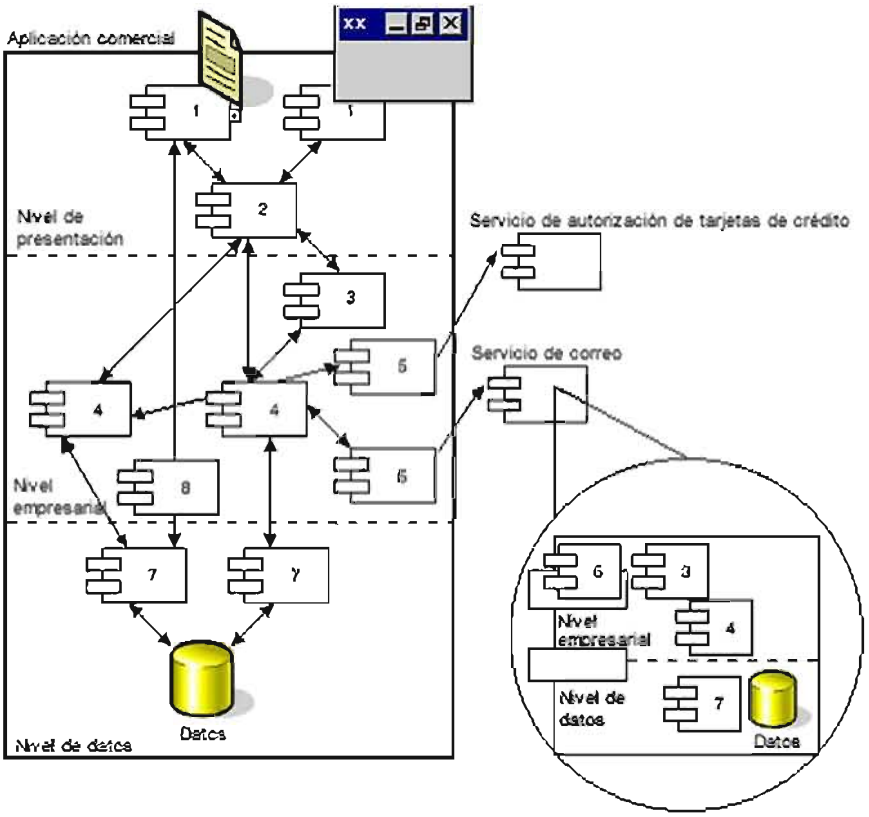
También tienen un impacto entre sí (por ejemplo, la directiva de seguridad determina algunas de las reglas en la directiva de comunicación y viceversa).

Escenario de ejemplo

Para ayudar a identificar los tipos frecuentes de componentes, esta guía describe una aplicación de ejemplo que utiliza servicios externos. Aunque esta guía se centra en un ejemplo concreto, las recomendaciones de diseño indicadas se aplican a la mayor parte de las aplicaciones distribuidas, independientemente del escenario empresarial real.

El escenario descrito en esta guía es una extensión de la aplicación comercial descrita anteriormente en este capítulo. En este escenario, una empresa de venta al por menor ofrece a sus clientes la posibilidad de solicitar productos a través de un sitio Web de comercio electrónico o por teléfono. Los usuarios de Internet pueden visitar el sitio Web de la compañía y seleccionar productos de un catálogo en línea. De forma alternativa, los clientes pueden solicitar productos de una catálogo de pedidos por correo mediante una llamada por teléfono a un representante de ventas, que indica los detalles del pedido a través de una aplicación basada en Microsoft Windows. Una vez finalizado un pedido, los detalles de la tarjeta de crédito del cliente se autorizan mediante un servicio de tarjetas de crédito externo y la entrega se organiza a través de un servicio de correo externo.

La solución propuesta para este escenario es un diseño basado en componentes compuesto por una serie de componentes, tal y como se muestra en la figura:



En la figura se muestra la aplicación comercial compuesta por varios componentes de software, que se agrupan en niveles lógicos según el tipo de funcionalidad que proporcionan. Observe que desde el punto de vista de la aplicación comercial, los servicios de autorización de tarjetas de crédito y de correo se pueden considerar componentes externos. Sin embargo, internamente los servicios se implementan más como las aplicaciones normales y contienen los mismos tipos de componentes (aunque los servicios de este escenario no contienen un nivel de presentación, sino que publican su funcionalidad a través de una interfaz de servicios mediante programación).

5. MODELO OSLO

La arquitectura orientada en servicio (SOA) debe evolucionar hacia un diseño más ágil, desarrollo, y una implementación el proceso. Sobre todo, debe cerrar la brecha en medio IT y el negocio.

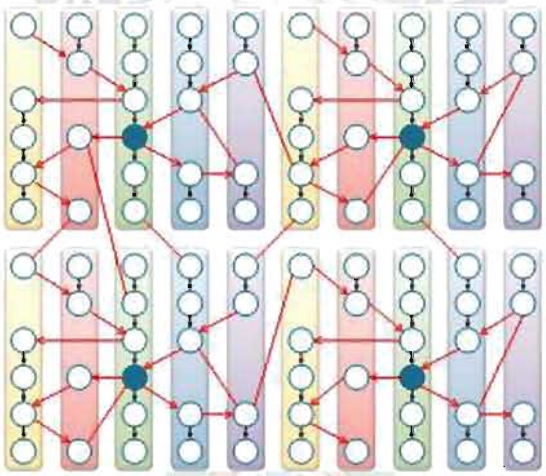
Introducccion

Un mapa entre teoría modeladora (El desarrollo conducido en modelo y SOA vuelta en modelo) y posible las implementaciones con la siguiente ola de Corporación Microsoft modelando Tecnología es "Oslo" Corporación Microsoft ha estado revelando mucha

información acerca de " Oslo " Al dar varias Exhibiciones Preliminares Comunales (CTPs) de Tecnología O las betas anticipadas. Sin embargo, la mayor parte de la información que está disponible "Oslo" es muy enfocado en tecnología (la interna " M " - el lenguaje Implementación).

Problema: La complejidad de SOA.

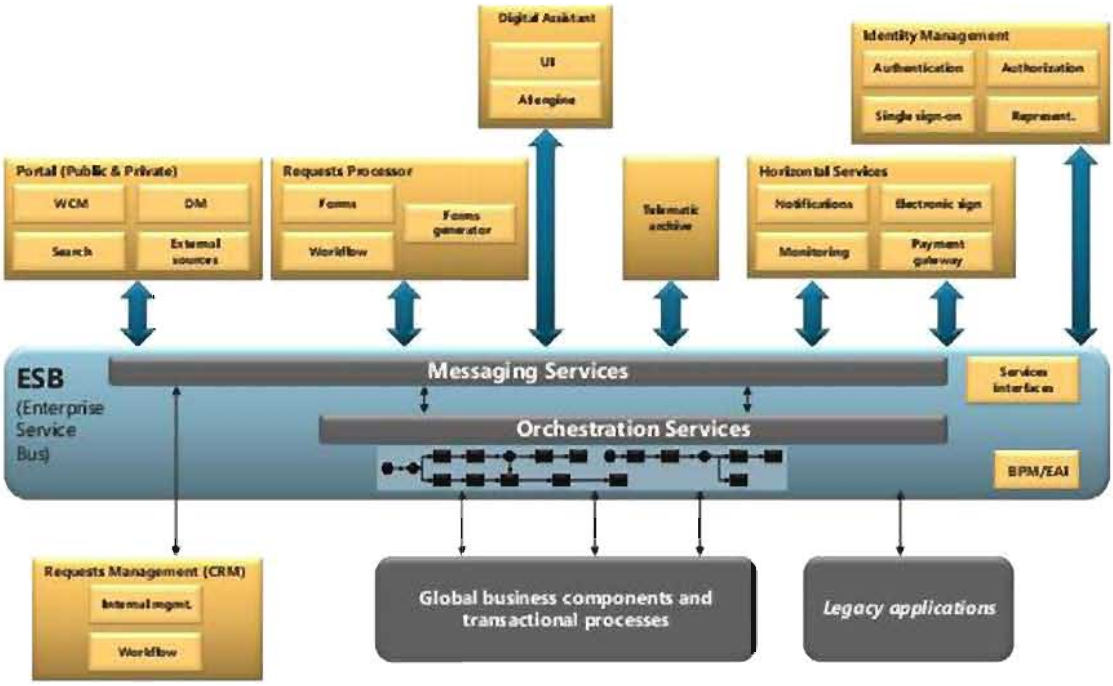
¿Cuál es el problema más importante en IT? ¿Es los lenguajes, las herramientas, los programadores? Bien, según investigadores y usuarios comerciales, es complejidad del software. Y esto ha sido el problema principal desde computadoras fuera nacido.
El desarrollo aplicativo es en realidad costoso el proceso, y los requisitos del software son crecientes. Integración, disponibilidad, fiabilidad, dimensionalidad, seguridad, e integridad /obediencia. Se están volviendo asuntos más complicados, del mismo modo que se pongan más críticos. La mayoría de las soluciones hoy requieren el uso de una colección de tecnologías, no sólo uno. Al mismo tiempo, en muchos aspectos, las aplicaciones de la empresa han evolucionado en algo que es también complejo a ser realmente efectivo y dinámico. Acerca de Arquitectura SOA, cuando una organización tiene Muchos asociaron servicios, la red lógica puede ser extremadamente difícil de manejar.



La anterior figura refleja dicha complejidad, cada círculo podría representar un servicio y cada caja el aplicativo en el que se encapsula.

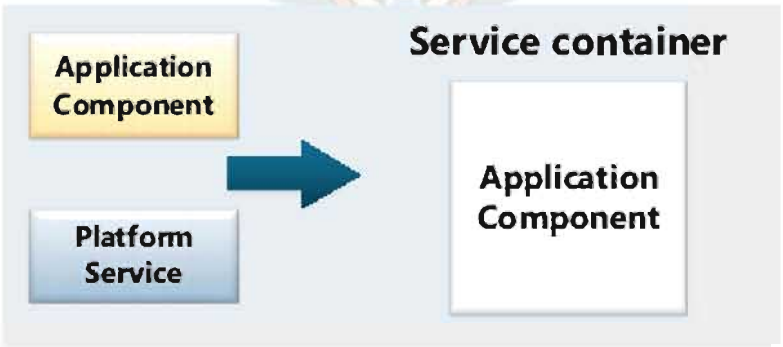
El problema mayor en esta red de servicios es que todos estos se conectan directamente, estos puntos de conexión llegar a ser difíciles de gestionar.

Una aproximación a una solución es el uso de un ESB, como se ve en la siguiente figura:



El Lenguaje Es la Interfaz (Cloudly Speaking)

En el desarrollo componente tradicional, la plataforma subyacente en si los servicios corren en el espacio de ejecución del componente; en un servicio la arquitectura orientada (SOA), sin embargo, el componente aplicativo es patrocinado en un envase de servicio. El envase de servicio provee las interfaces basadas en normas y los acuerdos (SLAs) que consideran la fiabilidad, y la disponibilidad.
Nosotros ahora requerimos la habilidad para que clientes controlen programáticamente los SLAs que son siempre dinámicas por el componente.
Esto es logrado fácilmente si la interfaz primaria del componente es un lenguaje de programación “, ” en lugar de un set de APIs. En otras palabras, los cambios entre componentes son scripts o programas, no las peticiones y las respuestas. En lugar de una unión o una selección de todo lo los objetos contenidos como sus elementos primarios de la interfaz, un componente debería tener como su interfaz primaria un intérprete de lenguaje que tiene uno.



La interfaz ahora consiste de: Las acciones o los métodos que se proveyó por el componente Objetos (la funcionalidad básica). La infraestructura y los servicios de la plataforma — como la tolerancia a la falla, reparación y conexión mancomunando — eso es provisto Por el armazón y constituya las medidas y estado.

¿SOA Realmente Soluciona la Complejidad, y de forma Agile?

Por otra parte, SOA ha sido la Tierra Prometida y una principal tecnología de la información. Hay muchos ejemplos en SOA en la cual la teoría es perfecta, pero su implementación no lo es. La realidad en eso hay problemas y obstáculos de más en la implementación de SOA. La interoperabilidad entre plataformas diferentes, así como también realmente el ser autónomo de servicios consumió de aplicaciones desconocidas, son realmente es dolores de cabeza y procesos lentos.
SOA prometió una separación de funcionalidad de un nivel más alto que la programación orientada a objetos y, especialmente, una **Mejor arquitectura desacoplada del componente /servicios que haría decrecer la complejidad externa.**

Pero esto es cierto? Las experiencias en varias organizaciones no fueron explícitamente las mejores. Una de las razones mayores es:

"En SOA, podemos tener un montón de servicios autónomos independientemente evolucionando — sin conocer quién va a usar mis servicios o cómo mis servicios van a ser consumidos — y esos servicios aun debería naturalmente estar conectado."

SOA como modelo: ¿Es Esa la Solución?

Finalmente, las organizaciones deben cerrar la brecha entre IT y el negocio mejorando la comunicación y la colaboración entre ellos.
La pregunta es, ¿" Quién tiene que surgir?" Probablemente, IT tiene que acercarse al negocio y sea bastante más asequible y acogedor.
Para apalancar una organización se tienen que manipular la tecnología, pero con un nuevo método, porque todavía se tiene que enfocar adelante el negocio; las personas que deseen modelar los servicios y procesos de negocio no tienen que aprender a programar un servicio o la aplicación utilizando un lenguaje de bajo nivel como C#, VB, o Java.
Algún día, SOA vuelta en modelo podría solucionar este problema. No sólo de orquestación
Desde el punto de vista — uno que está más cerca para el lado comercial.
SOA vuelta en modelo tendrá un montón de ventajas, aunque tenga que afrontar muchos retos. Pero la meta es solucionar más los problemas típicos de SOA:

- El modelo es el código. Ni la recopilación ni la traducción debería ser menester. Ésta es una gran ventaja sobre código directamente.

- Las soluciones que SOA vuelta en modelo crea serán transigentes con principios SOA, porque todavía confiamos en SOA. Qué somos cambiantes es sólo la forma en la cual crearemos y orqueste servicios.
- En un momento, habremos logrado nuestra meta: Para cerrar la brecha en medio de IT y el negocio, entre informático en jefe y director general.

DISEÑO E IMPLEMENTACION

1. CONTEXTO GENERAL DE LA SOLUCIÓN

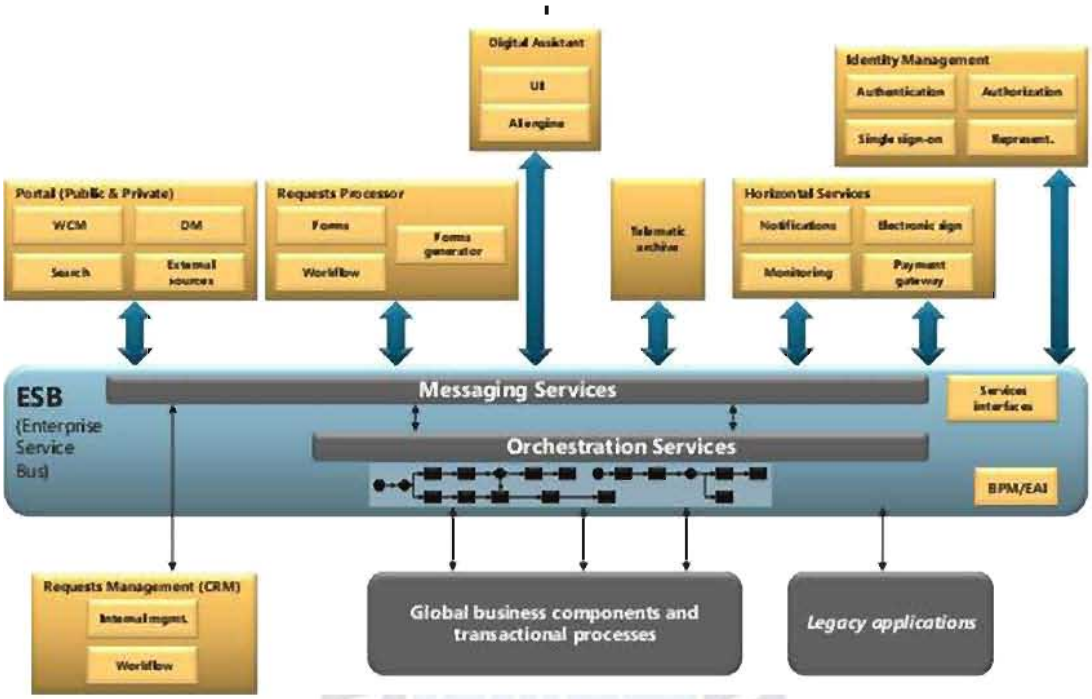
Actualmente la integración con los proveedores mayúsculos de servicios de los aplicativos de una Institución es heterogénea, por la granularizacion de servicios que presenta cada uno por separado. Por lo cual un cambio de proveedores afectara de manera drástica el buen funcionamiento de todos los aplicativos involucrados, para poder hacer factible este cambio se debe proporcionar una solución orientada a SOA, basándonos en los siguientes aspectos:

- Continuidad en los procesos actuales de negocio, de forma independiente a las interfaces inyectadas para consumir data de los nuevos proveedores.
- Gestión de servicios de los proveedores, para el consumo de aplicativos macros, esta gestión debe cubrir los siguientes aspectos:
    - Priorización, ordenación y sincronización de las peticiones hacia proveedor.
    - Contemplar la integridad transaccional y el manejo de errores.
    - Dotar de interfaces de monitoreo.
    - Continuidad de los procesos de negocio de forma transparente.
- Construcción de las interfaces de acceso a los aplicativos.
- Construcción de transformadores para factorizar las estructuras manejadas en el intercambio de datos Proveedor y Aplicativos.

Panorama general "Bus de Servicio Empresarial"

De forma general es se busca que a través del BUS de servicio empresarial se implemente el área de intercambio de información entre los distintos aplicativos que se encuentran dentro de una institución.

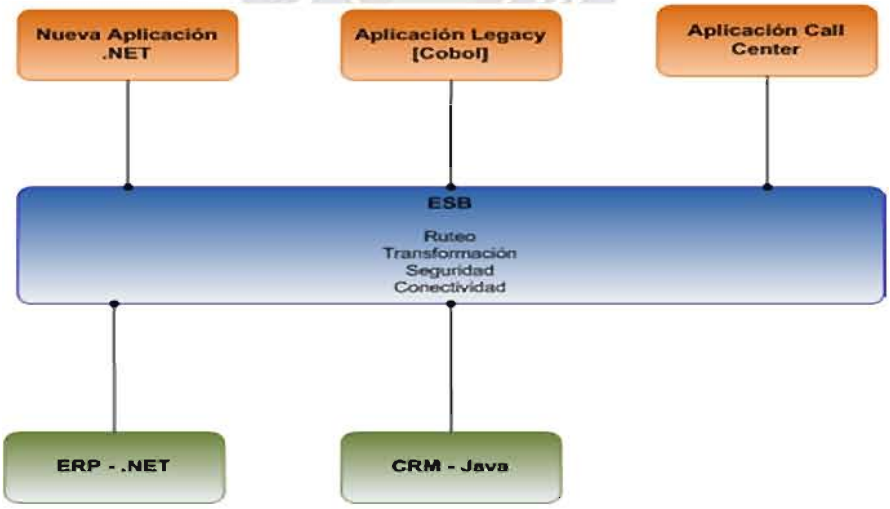
El intercambio de la mensajería de forma orquestada a través del BUS permite abstraer el flujo comercial de los procesos operativos de la organización, alimentándolos con información en línea de todas los repositorios (servicios).



Especificación Arquitectónica:

Para poder lograr construir este gestor es necesario estudiar los componentes básico que requerimos construir tanto para el intercambio de mensajería como de la orquestación:

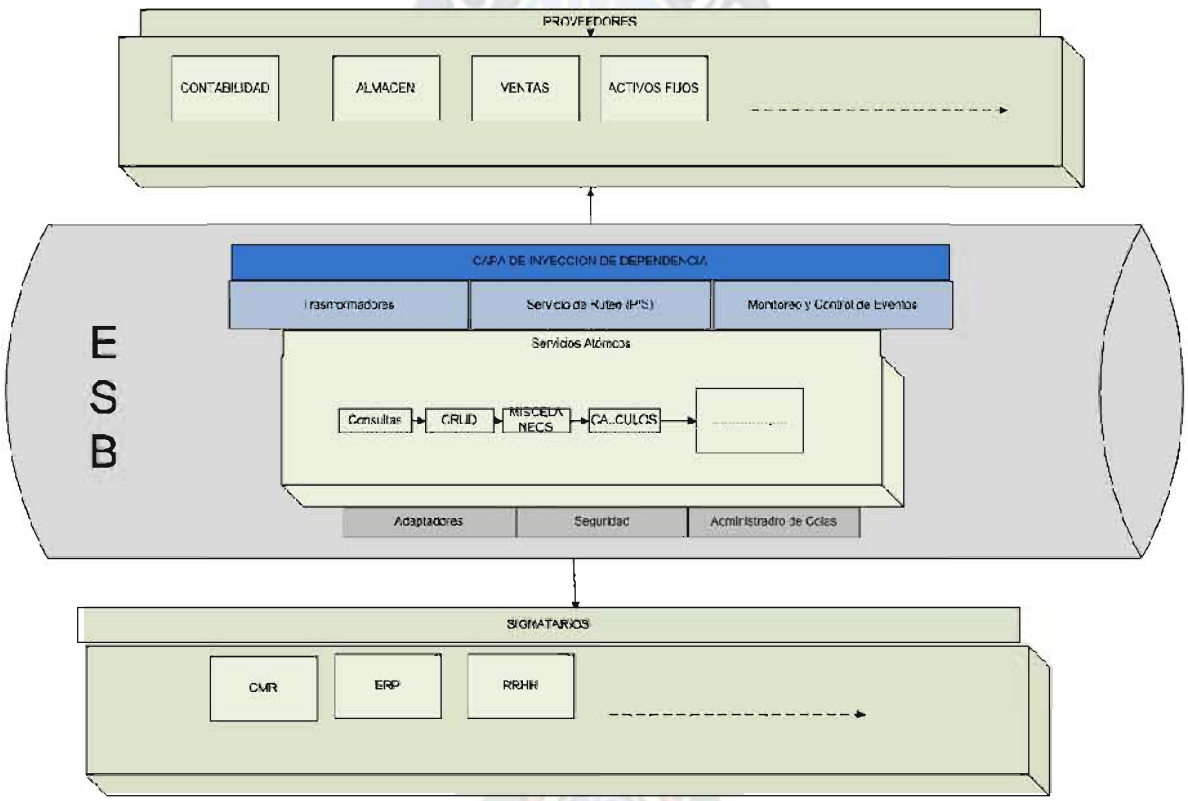
Vista inicial de mensajería mensajería:



Dentro del alcance técnico se plantea:

- Comunicación con servicios que estén expuestos en distintos protocolos, esta tarea debe ser designada a un componente que siga el patrón command, al mismo dentro de la especificación de la denominaremos adaptador.
- Transformación de datos, el valor agregado de la data alimentada con información de distintos repositorios será designado al componente transformador.
- La comunicación inteligente y orquestación del proceso será realizada por el componente canal.

Para poder cubrir todos los requerimientos definidos, es necesario construir la siguiente estructura:



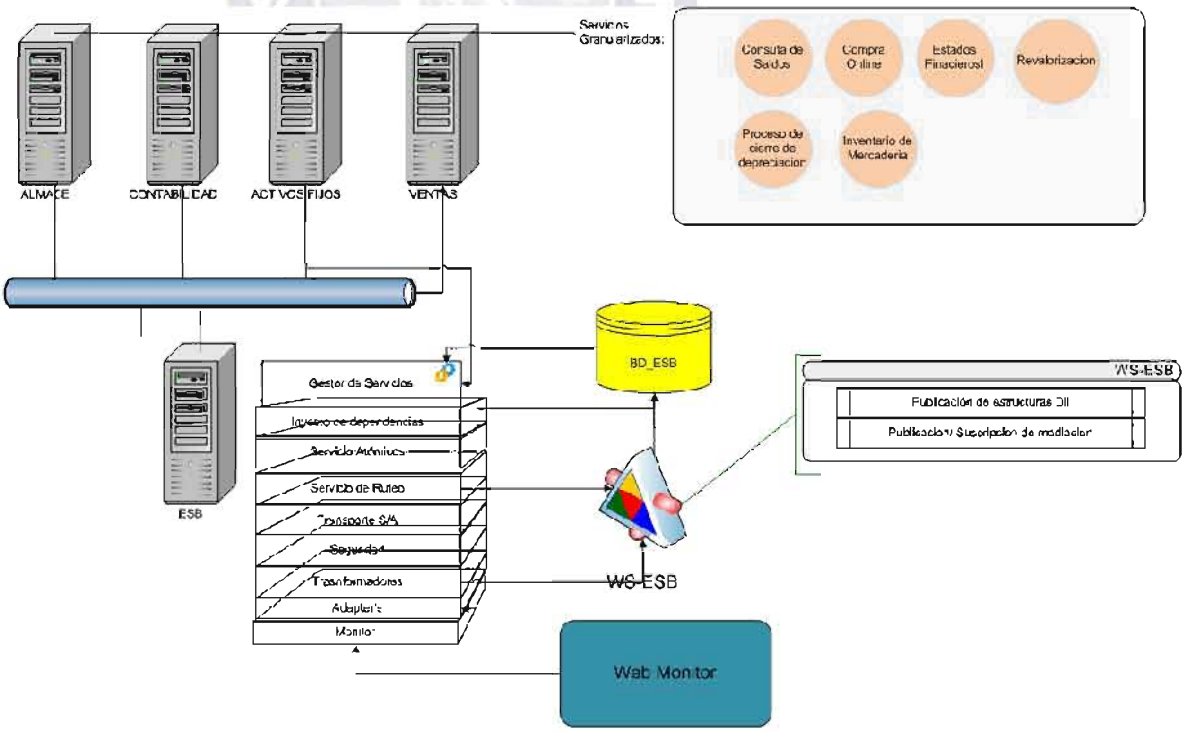
1. Servicios Macros: Conjunto de aplicativos Macros que consumirán datos del proveedor.
2. ESB (Bus de servicio empresarial): La definición de capas de la arquitectura del Bus se explica:
    a. Capa de Inyección de dependencia: para cada aplicativo propio del consumidor, el proveedor se implementara librería basa en los contratos estructurados para el consumo de los recursos del proveedor.
    b. Transformadores: Componente que permitirán convertir los documentos XML en objetos serializados que requieran los aplicativos macro. La funcionalidad de los mismos pueden extenderse a la mejora del mensaje (Agregación de información

faltante al paquete en base a parámetros de entradas proporcionadas por el consumidor).

c. Servicio de ruteo: Comprende el ruteo en base a los parámetros: origen, destino y prioridad; además de control una bitácora donde se registra todas las peticiones. Las mediaciones ser realizaran mediante el método de publicación / suscripción, bajo el criterio de un proxy.
d. Monitoreo y control de eventos: presentara todas las peticiones en cola y procesadas, por el ESB, el mismo debe ser extensible a manejar indicadores respecto a los mismos.
e. Servicios atómicos: Conjunto de servicios otorgados por el proveedor, los mismos deben ser registrados en un BD relacional para poder invocarlos de forma dinámica.
f. Adaptador: interfaz con el proveedor que proporcionará: comunicación (protocolos) y lógica de autenticación contra el proveedor.
g. Seguridad: Autenticación y encriptación de los datos enviados al proveedor. Transporte síncrono: manejo encolado de mensajes, proveedor de indicadores al monitor.

3. Proveedores: Conjunto de servicios externos, actualmente el único proveedor es ATC.

El esquema de dominio es:



Los componentes de dominio se detallan:

- Servicios Granularizados: Son los servicios que otorgan los sistemas macros de una organización a los usuario finales.
- Servicios provistos: son los servicios provistos por un proveedor sea interno o externo.
- Contabilidad, RRH, etc. son los sistemas macros del BCP.
- ESB: será la infraestructura que data soporte a la gestión de servicios entre los sistemas macros y los proveedores.
- Gestor de servicios: Arquitectura interna del ESB.
- BD_ESB: repositorio propio del ESB.
- WS ESB: Webservices proveedor de las suscripción de servicios, y estructuras para los transformadores.
- Web Monitor: sitio web para la administración del ESB.
- VPN: canal dedicado al ESB y el resto de la infraestructura.

2. CONTEXTO TECNOLÓGICO

Para poder construir los componentes mencionados, requerimos de una plataforma que nos otorgue soporte de intercomunicabilidad de protocolos y además de robustez en las herramientas de intercomunicación.

Dentro de la construcción definiremos como framework base WCF. A continuación una descripción breve de esta extensión de .Net Framework:

Fundamentos de WCF

Un Servicio WCF es un programa que expone una colección de '*Endpoints*' (extremos o puntos de entrada de WCF). Cada '*Endpoint*' es un punto de entrada de comunicación 'con el mundo'. Un Cliente es un programa que intercambia mensajes con uno o mas '*Endpoints*'. Un Cliente puede exponer también un '*Endpoint*' para recibir mensajes de un Servicio basado en un patrón de intercambio de mensajes de tipo *dúplex*.
Las siguientes secciones describen estos fundamentos en más detalle.

Endpoints

Un 'Endpoint' de un Servicio está compuesto por una '*Address*' (Dirección), un '*Binding*' (Enlace) y un '*Contract*' (Contrato).

La dirección de un 'Endpoint' es una dirección de red *donde* reside dicho 'Endpoint'. La clase **EndpointAddress**' epresenta una dirección de un 'Endpoint' WCF.

El 'binding' de un 'Endpoint' especifica *como* se comunica dicho 'Endpoint' con el resto del mundo, incluyendo aspectos como el protocolo de transporte (p.e. TCP, http, etc.), tipo de codificación (p.e. texto, binario), y requerimientos de seguridad (p.e. SSL, seguridad basada en mensajes SOAP, etc.). La clase **Binding** representa un *binding* de WCF.

El contrato de un *endpoint* especifica *qué* comunica dicho *endpoint* y básicamente está compuesto por una colección de mensajes organizados internamente en operaciones que tienen un patrón de intercambio de mensajes (Message Exchange Patterns ó MEPs), como '*one-way*' (un sentido), *dúplex* y *request/reply* (petición/respuesta). La clase **ContractDescription** representa un contrato WCF.

La clase **ServiceEndpoint** representa un *Endpoint* y tiene un **EndpointAddress**, un **Binding** y un **ContractDescription** que corresponden a la dirección del *endpoint*, al enlace y al contrato, respectivamente (ver figura 1).

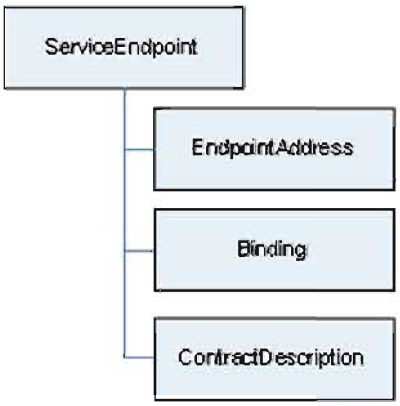


Figura 1. Cada Endpoint del Servicio contiene un Binding y un Contrato representado por 'ContractDescription'.

EndPointAddress

Un EndPointAddress es básicamente un URI, un identificador y una colección de cabeceras opcionales, como muestra la Figura 2.

La identidad de seguridad de un EndPoint es normalmente su propio URI; sin embargo, en escenarios avanzados la identidad puede establecerse explícitamente de forma independiente del URI haciendo uso de la propiedad de la dirección '**Identitiy**'.

Las cabeceras opcionales se utilizan para proporcionar información adicional de direccionamiento, mas allá del URI del EndPoint. Por ejemplo, las cabeceras de dirección son útiles para diferenciar entre varios Endpoints que comparten la misma dirección URI.

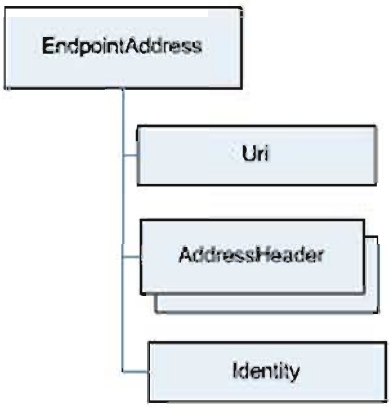


Figura 2. EndPointAddress contiene un URI y AddressProperties contiene una Identidad y una colección de cabeceras de dirección.

Bindings

Un Binding está compuesto por un nombre, un '*namespace*' (espacio de nombres de programación, es decir, 'su apellido'), y una colección de elementos binding compuestos (figura 3). El nombre y *namespace* del *binding* lo identifican de forma única en los metadatos del servicio. Cada elemento del binding describe un aspecto de *cómo* se comunica el EndPoint con el 'resto del mundo'.

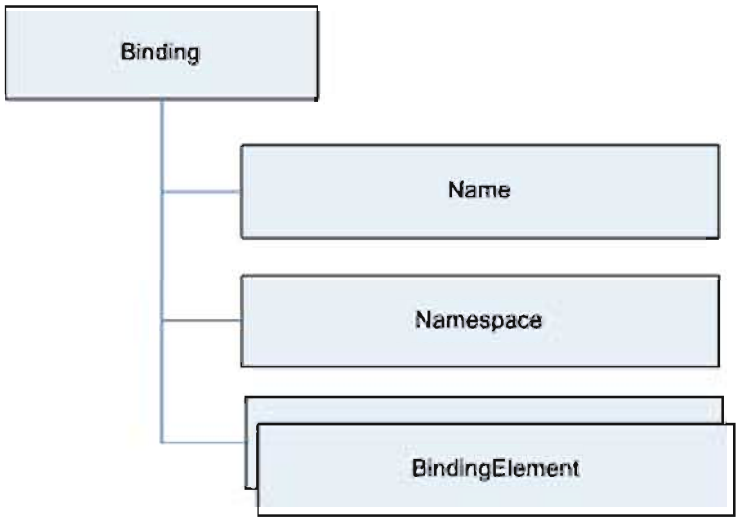


Figura 3. La clase Binding y sus componentes

Por ejemplo, la **Figura 4** muestra una colección de elementos de *binding* que contiene tres elementos de *binding*. La presencia de cada elemento de *binding* describe parte del *como* comunicarse con el *EndPoint*. El **TcpTransportBindingElement** indica que el EndPoint se comunicará con el resto del mundo usando TCP como protocolo de transporte. **ReliableSessionBindingElement** indica que el *EndPoint* hace uso de mensajería confiable para

proporcionar aseguramientos de entrega de mensajes. **SecurityBindingElement** indica que el *EndPoint* hace uso de seguridad basada en mensajes SOAP. Cada elemento del *binding* normalmente tiene propiedades que describen aun en más detalle el *cómo* comunicarse con el *EndPoint*. Por ejemplo, el **ReliableSessionBindingElement** tiene una propiedad llamada '**Assurances**' que especifica los requerimientos de aseguramientos de entrega de mensajes, como ninguno, uno como mínimo, uno como máximo, o exactamente uno.

Figura 4. Un Binding ejemplo con tres elementos de binding

El orden y los tipos de los elementos de *binding* en los *Bindings*, es importante: La colección de elementos de *binding* se utiliza para construir una pila ordenada de comunicaciones de acuerdo al orden de los elementos de *binding* en la colección de elementos de *binding*. El último elemento de *binding* a añadir a la colección corresponde al componente del fondo de la pila de comunicaciones, mientras que el primero corresponde al componente de arriba. Los mensajes de entrada fluyen a través de la pila desde el fondo hacia arriba, mientras que los mensajes de salida fluyen desde arriba de la pila hacia abajo. Por lo tanto, el orden de los elementos de *binding* en la colección afecta directamente el orden en el cual los componentes de la pila de comunicaciones procesan los mensajes. Es importante destacar que WCF proporciona un conjunto de *bindings* pre-definidos que pueden utilizarse en la mayoría de los escenarios, pero siempre se tiene la opción de implementarse *bindings* 'custom' o propios.

Contratos

Un contrato WCF es una colección de operaciones que especifican *qué* comunica el *EndPoint* al resto del mundo. Cada operación es un simple intercambio de mensaje, por ejemplo un intercambio de mensaje '*one-way*' o '*request/reply*'.

La clase **ContractDescription** se utiliza para describir contratos WCF y sus operaciones. Dentro de un ContractDescription, cada operación de contrato tiene una OperationDescription correspondiente, que describe aspectos de la operación como si la operación es 'one-way' o 'request-reply'. Cada OperationDescription también describe los mensajes que conforman la operación usando una colección de **MessageDescriptions**.

Un ContractDescription se crea normalmente basándonos en un interfaz o clase que define el Contrato, haciendo uso del modelo de programanción de WCF. Este tipo se anota con el atributo

ServiceContractAttribute y sus métodos que corresponden a operaciones de EndPoint, se anotan con el atributo **OperationContractAttribute**. También se puede construir 'a mano' un **ContractDescription** sin comenzar con un tipo CLR marcado con atributos.

Un contrato *Duplex* define dos conjuntos lógicos de operaciones: Un conjunto que expone el servicio para que el Cliente llame o invoque y un conjunto que el Cliente expone para que el Servicio llame o invoque. El modelo de programación para definir un contrato dúplex consiste en dividir cada conjunto anterior en un tipo separado (cada tipo debe ser un interfaz o una clase) y marcar el contrato que representa a las operaciones del servicio con ServiceContractAttribute, referenciando al contrato que define las operaciones cliente (ó *callback*). Además, ConmtractDescription contiene referencias a cada uno de los tipos y los agrupa en un contrato dúplex.

De forma similar a los *Bindings*, cada contrato tiene un nombre y un namespace que lo identifican de forma única en los metadatos del Servicio.
Cada contrato tiene también una colección de ContractBehaviors que son módulos que modifican o extienden los comportamientos del contrato. La siguiente sección cubre dichos 'behaviors' (comportamientos) en mas detalle.

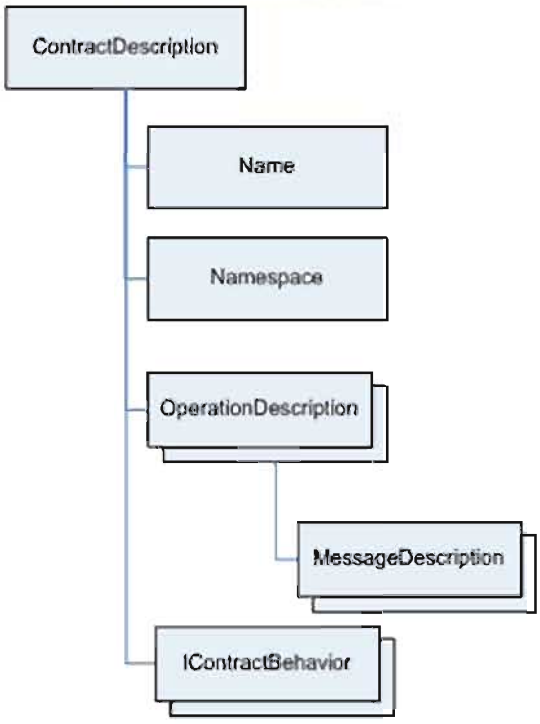


Figura 5. La clase ContractDescription describe un contrato WCF

Behaviors (Comportamientos)

Los *Behaviors* son tipos que modifican o extienden la funcionalidad del Servicio o del Cliente. Por ejemplo, los comportamientos de metadatos que implementa ServiceMetadataBehavior controla si el servicio publica metadatos. De forma similar, los comportamientos de seguridad controlan

aspectos de impersonación y autorización, mientras que el comportamiento de transacciones controla enlistamientos completados automáticos de transacciones (Auto-Complete).

Los comportamientos también participan en el proceso de construcción del canal (*channel*) y pueden modificar para que el canal se base en características especificadas por el usuario y/u otros aspectos del Servicio o Canal.

Un comportamiento de Servicio es un tipo que implementa **IServiceBehavior** y se aplica a los Servicios. De forma similar, un comportamiento de canal (*Channel Behavior*) es un tipo que implementa **IChannelBehavior** y se aplica a canales Cliente.

Descripciones de Servicio y Canal

La clase **ServiceDescription** es una estructura en memoria que describe un Servicio WCF incluyendo los *EndPoints* expuestos por el Servicio, los Behaviors aplicados al servicio y el tipo (una clase) que implementa el servicio (ver figura 6). ServiceDescription se utiliza para crear metadatos, codificar/configurar, y canales.Se puede construir a mano este objeto *ServiceDescription.* También se puede construir a partir de un tipo marcado con ciertos atributos WCF, que es el escenario mas común. El código de este tipo puede escribirse a mano o generado a partir de un documento WSDL mediante la utilidad **svcutil.exe**.
Aunque los objetos ServiceDescription pueden crearse e instanciarse explícitamente, muchas veces se crean 'detrás del telón' como parte de un Servicio en ejecución.

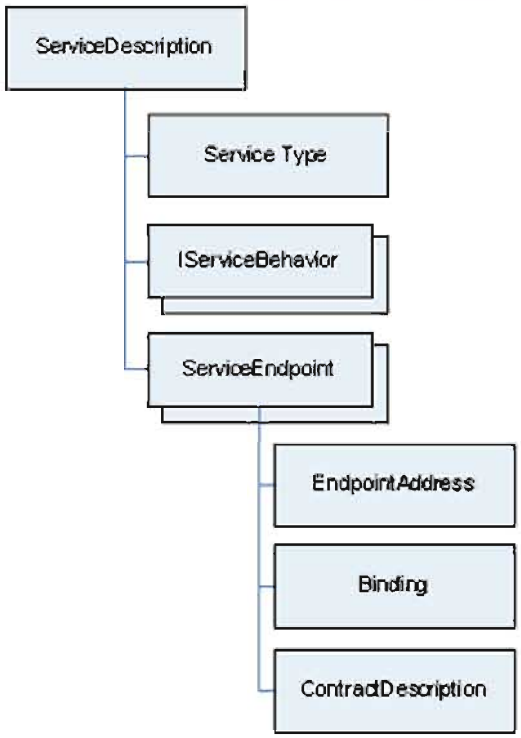


Figura 6. Modelo de objetos ServiceDescription

Igualmente en el lado cliente, un ChannelDescription describe un canal cliente WCF para un endpoint específico (Figura 7). La clase **ChannelDescription** tiene una colección de IChannelBehaviors, que son comportamientos aplicados al canal. También tiene un ServiceEndPoint que describe el EndPoint con el cual el canal se comunicará. Es importante destacar que, al contrario que ServiceDescription, ChannelDescription contiene solamente un ServiceEndPoint que representa el EndPoint objetivo con el cual se comunicará el canal.

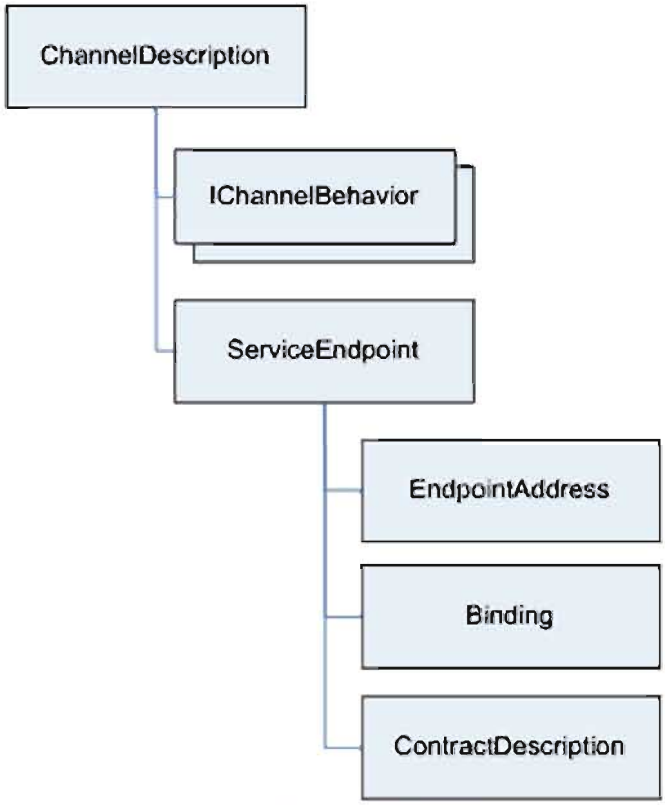


Figura 7. Modelo de objetos de ChannelDescription

Runtime de WCF

El *runtime* de WCF es el conjunto de objetos responsable del envío y recepción de mensajes. Por ejemplo, cosas como formato de mensajes, aplicación de seguridad, y transmitir y recibir mensajes mediante varios protocolos de transporte, así como proporcionar mensajes recibidos a la operación apropiada, todo corresponde al runtime de WCF. Las siguientes secciones explican los conceptos clave del runtime de WCF.

Mensaje

El mensaje WCF es la unidad de intercambio de datos entre un Cliente y un Endpoint. Un mensaje es básicamente una representación en memoria de un ***InfoSet*** mensaje SOAP. Hay que destacar que un mensaje (en lo relatico a su comunicación por la red) no está ligado a texto XML. Por el

contrario, dependiendo de qué mecanismo de '*encoding*' se utilice, un mensaje puede serializarse basado en el formato binario de WCF, texto XML o cualquier otro formato propio.

Channels (Canales)

Los canales son la abstracción principal para mandar mensajes y recibir mensajes desde un Endpoint. Básicamente hay dos tipos de canales: 'Canales de Transporte' que gestionan el envío y recepción de *streams* de 'octetos opacos' basándose en algún tipo de protocolo de transporte como TCP, UDP o MSMQ.
Los canales de protocolo, por otro lado, implementan un protocolo basado en SOAP para procesar y posiblemente modificar mensajes. Por ejemplo, el canal de seguridad añade y procesa cabeceras de mensajes SOAP y puede modificar el cuerpo del mensaje, por ejemplo, cifrándolo (encriptándolo).
Los canales son componentizables de forma que un canal puede estar basado sobre otro canal que a su vez está basado en un tercer canal.

EndPointListener

Un *EndPointListener* es el *runtime* equivalente a un *ServiceEndpoint*. El *EndPointAddress*, *Contrato* y *Binding* de un *ServiceEndpoint* (representando el *donde*, *qué* y *como*), corresponde a la dirección de escucha del *EndpointListener*, filtrado de mensajes, entrega, y pila de canales, respectivamente. El *EndPointListener* contiene la pila de canales que es responsable del envío y recepción de mensajes.

ServiceHost y ChannelFactory

El runtime del Servicio WCF se crea normalmente 'detrás del telón' llamando a ServiceHost.Open(). ServiceHost (figura 6) dirige la creación de un ServiceDescription a partir del tipo del servicio e instancia la colección ServiceEndpoint del ServiceDescription con EndPoints definidos en el .config o código o en ambos. ServiceHost entonces hace uso de ServiceDescription para crear la pila de canales en la forma de un objeto EndPointListener para cada ServiceEnpoint en el ServiceDescription.

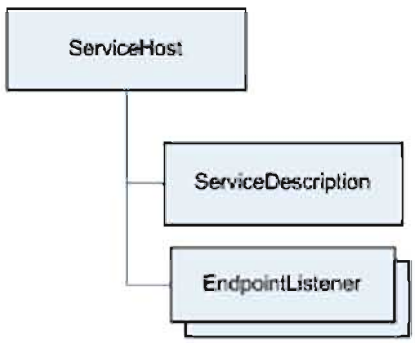


Figura 8. Modelo de objetos ServiceHost

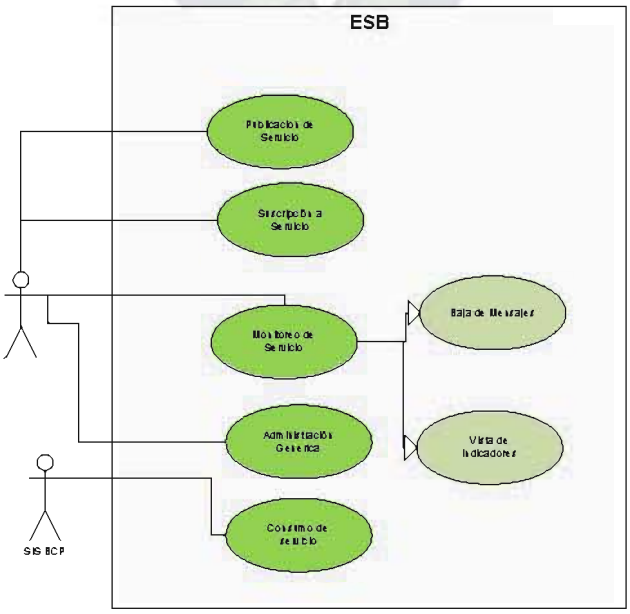
De forma paralela, en el lado cliente, el *runtime* cliente es creado por **ChannelFactory**, que es el equivalente del cliente a *ServiceHost*.
ChannelFactory dirige la creación de un *ChannelDescription* basado en un tipo de contrato, un *binding*, y un *EndPointAddress*. Entonces hace uso de este *ChannelDescription* para crear la pila de canales Cliente.
De forma contraria al *runtime* del Servicio, el *runtime* cliente no contiene *EndPointListeners* porque un Cliente siempre inicia la conexión hacia el Servicio, así que no necesita estar 'escuchando' para aceptar conexiones de entrada.

Resumen

Los servicios WCF exponen una colección de Endpoints donde cada Endpoint es una puerta de comunicación hacia el resto del mundo. Cada Endpoint tiene una *Address*, un *Binding* y un *Contract* (ABC). La dirección es *donde* reside el Endpoint, el *Binding* es *como* se comunica el *Endpoint*, y el *Contract* es *qué* ofrece/comunica el Endpoint. En el servicio, una ServiceDescription contiene la colección de ServiceEndpoints, cada uno describiendo un Endpoint que expone el servicio. Desde esta descripción, ServiceHost crea un runtime que contiene un EndpointListener para cada ServiceEndPoint en el ServiceDescription. La dirección, binding y contrato del Endpoint (representando el cómo, qué y cómo) corresponden a las direcciones de escucha del EndpointListener, filtro y despacho de mensajes, y pila de canales, respectivamente.
De forma similar, en el Cliente, un ChannelDescription contiene el ServiceEndpoint con el cual el cliente se comunica. Desde este ChannelDescription, ChannelFactory crea la pila de canales que se comunican con los Endpoints del Servicio.

3. DESCRIPCIÓN DEL PROCESO DE ANÁLISIS

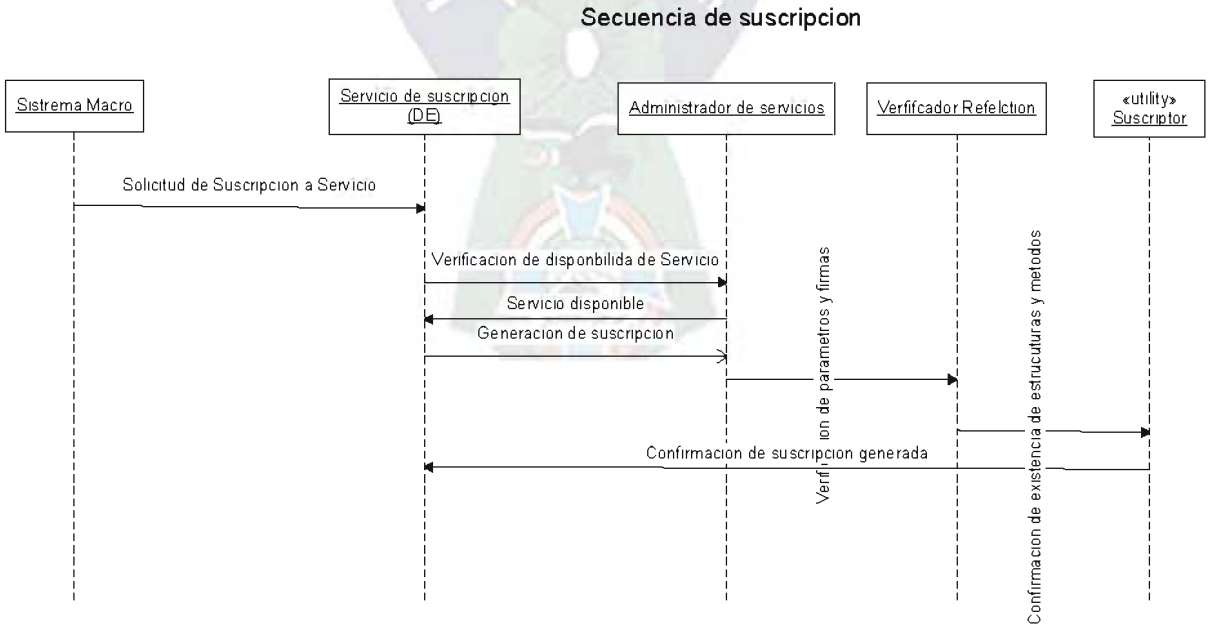
El ESB como gestor de servicios, debe proporcionar las siguientes funcionalidades:



- Publicación de Servicios comprende el proceso de realizar un deploy en caliente (cuando el servicio está funcionando) de un servicio que prestara algún proveedor. El proceso comprende desde el registro del servicio en la bd hasta el cargado de la Dll al servidor de aplicaciones.
- Suscripción del servicio comprende el proceso de abrir un nuevo canal desde el registro del proceso que consume el servicio al cual se desea suscribirse, evaluación de prioridades en la cola, ruteo en caso de usar un servicio intermedio para sacar la data del servicio final a consumir y el registro de la Dll que contendrá el algoritmo para la transformación de los datos iníciales al input requerido por los servicios de los proveedores.
- Monitoreo de Servicio: este módulo constará de dos partes:
    - Bajas de Mensajes: el cual estará disponible para mensajes disponibles en la cola, antes de su envió al servicio proveedor.
    - Vista de indicadores, presentara el tiempo de espera del mensaje, la prioridad asignada y la posición en la cola.
- Administración Genérica: Este modulo comprenderá el registro los parámetros generales del ESB, además del registro de los proveedores y otras entidades macros provenientes del diseño.
- Consumo de servicio: este modulo estará a cargo de exponer un servicio a cualquier signatario ya antes suscrito, dicha exposición se realizara mediante un canal.

El resto de los componentes, que no se encuentran no participan del flujo comercial; sin embargo estos serán detallados en el siguiente punto.

Los flujos correspondientes identificados son:



Descripción de Secuencia: la secuencia de suscripción describe el flujo mediante el cual un servicio macro se suscribe a un servicio que presta un proveedor. El paso de mensajes

1. Un sistema macro envía una solicitud al operador del ESB.
2. La solicitud es ingresada la Data Entry por el operador.
3. El administrador de servicios (componente que gestiona los servicios atómicos que presta los proveedores) verifica que el mismo esté disponible y manda la confirmación al DE.
4. El operador genera la suscripción,.
5. El administrador valida el cargado de las librerías dinamicas (DLL) mediante relfection (componente ofrecido por .NetFramework), una vez verificado el mismo es confinado a custodia de la capa de inyección de dependencia.
6. El administrador de servicios delega el registro de la suscripción al Suscriptor (Objeto de negocio que contiene los métodos de persistencia).
7. El suscriptor confirma la suscripción, después de haber creado un nuevo canal.

Delegación de tareas:

Servicio de suscripción DE: Interfaz del Data Entry que administrara todos los registros de suscripción a servicios, en base a los parámetros de: disponibilidad de servicios, manejo de canales existentes y parametrizacion del paquete (para su administración en colas).

Administrado de servicios: Realizara la gestión completa de los servicios granulares prestados por los proveedores. La información que genere será vista por el DE.

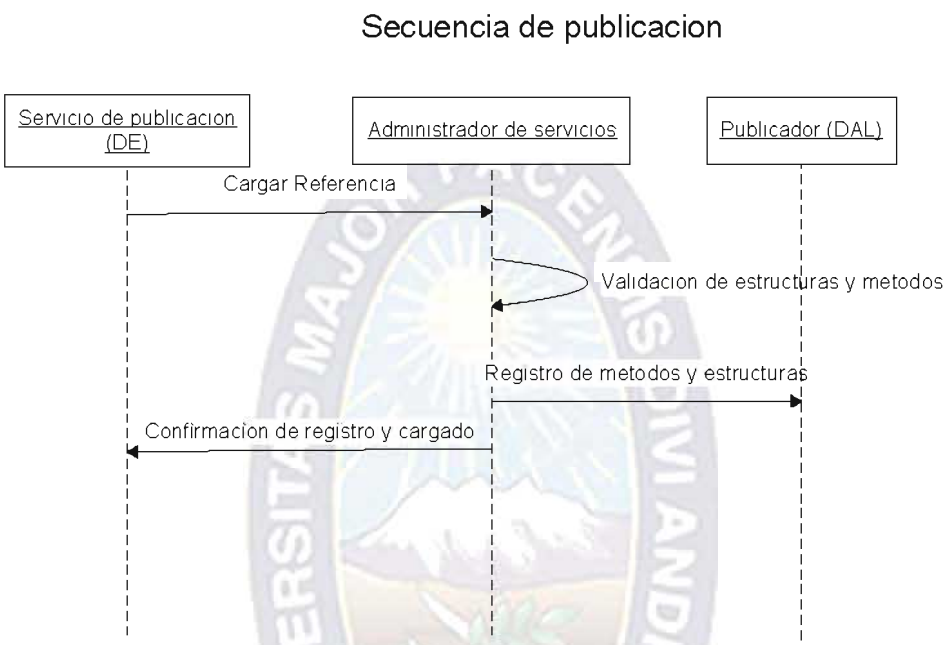
Capa de Inyección de dependencia: este componente no se encuentra en el grafico de secuencia, sin embargo el administrador de servicios delega la ejecución de las librerías cargadas a este componente.

Se debe tomar en cuenta que la librería cargada debe contemplar las siguientes características:

- Debe contener los algoritmos y la configuración necesaria para procesar los datos que ingresen y convertirlos en las entradas para los servicios a consumir del proveedor.
- Debe contener estructuras en base a los contratos definidos para poder instanciarlos de manera correcta.

Verificador reflexión: Componente del .NetFramewok que permite cargar las librerías y verificar las estructuras que contenga.

Suscriptor: componente objeto del negocio que contiene los métodos de persistencia correspondientes a un CRUD común.



Descripción de Secuencia: la secuencia de publicación, describe como agregar un servicio por parte del proveedor. El paso de mensajes se describe de la siguiente manera:

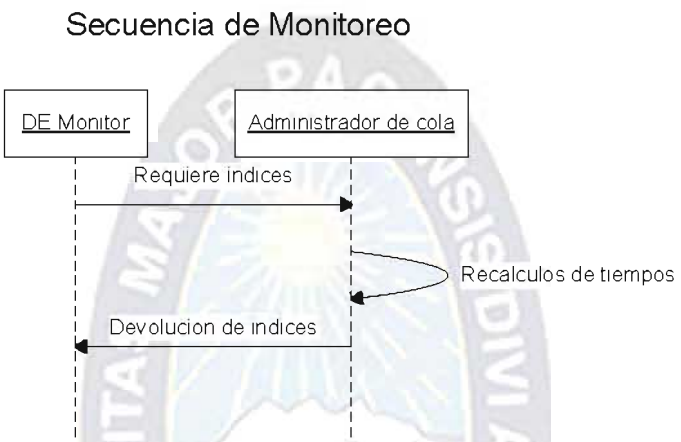
1. Servicio de Publicación que es una interfaz del DE manda la referencia del servicio a consumir del proveedor.
2. Administración de servicios valida el tipo de servicio y las operaciones correspondientes.
3. Manda la solicitud del registro de métodos y estructuras del mismo.
4. Publicador registra el servicio.
5. Administrador eleva a memoria el servicio y confirma el registro.

Delegación de tareas: la delegación de tareas se describe:

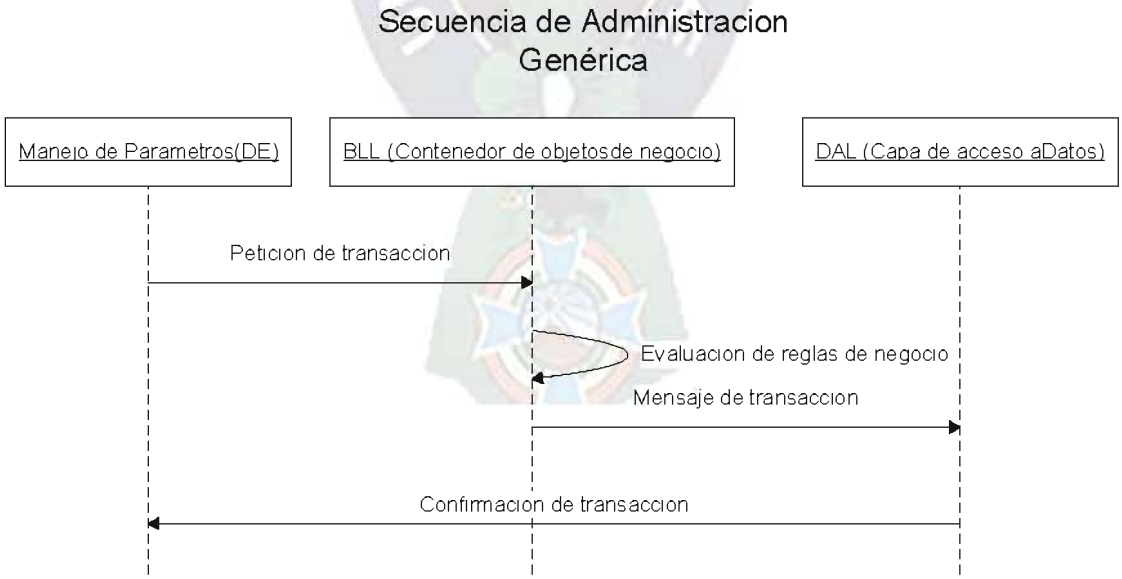
Servicio de publicación(DE): interfaz del data entry que permite el registro de nuevos servicios.

Administrador de servicios: componente que gestiona todos los servicios atómicos de los proveedores.

Publicador: componente de la capa de negocios que contiene los métodos de persistencia de una publicación.



Delegación de tareas:



Descripción de Secuencia: la secuencia de administacion genérica, es el flujo a seguir para realizar el registro de todos lo parametros y/o entidades genéricas para realizar clasificaciones sobre los servicios.

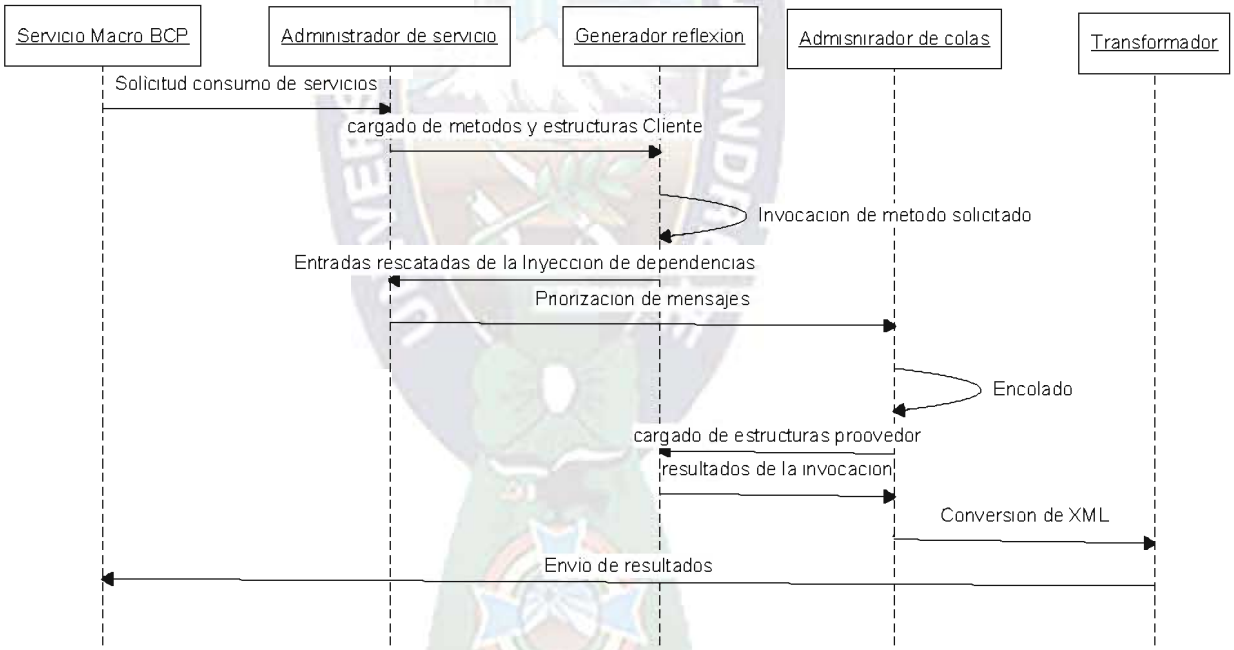
Los pasos en la secuencia se describen:

- El administrador de parametros, es una interfaz del Data Entry que permitre acceder al operador a todos los parámetros y entidades genéricas, atravez del mismo se en envia peticiones de transacciones al contenedor de objetos del negocio.
- El contenedor de objetos del negocio evalua las reglas correpondientes antes de envias el mensaje de proceder con la transaccion a la capa de acceso a datos.
- La capa de acceso a datos envia la confirmacion de transaccion procesada.

Delegación de tareas: la delegación de tareas se describe:

Administrador de parametros: interfaz del DE para visualizar y realizar transacciones con los parámetros y entidades genéricas.
Contenedor de objetos de negocio: contenedor logico de todos los objetos del negocio.
Capa de acceso a datos: contenedor de las clases genéricas para acceder a los datos.

SECUENCIA CONSUMO DE SERVICIO



Descripción de Secuencia: la secuencia de consumo de servicio presenta el consumo de un servicio por parte de un sistema macro ya suscrito.
La secuencia se explica de la siguiente manera:

- El sistema macro solicita el consumo de un servicio la cual ya fue suscrito.
- Se ejecuta y se carga en memoria los metodos de las librerias de la capa de inyeccion de dependencia mediante el generador relexion; se prepara el paquete y se lo pasa a la cola para su posterior ejecucion.

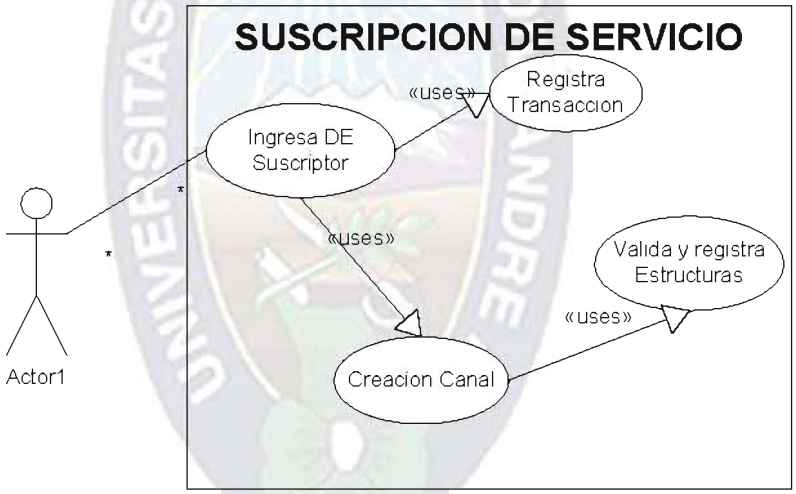
- El administrador de colas lo prepara con las prioridades correspondientes y lo coloca en una estructura tipo cola para su ejecucion.
- Transformadores: una ves ya obtenida la salida el transformador rescata el xml y lo convierte en un objeto que se configuro en la Dll del solicitante (la misma fue configurada durante la suscripcion del sistema macros)

Elaboracion del módulo suscriptor.

El componente suscriptor sera quien preste los servicios de:

- Creacion de un nuevo canal de comunicación.
- Realizar el deploy de librerias dinamicas en caliente.
- Registro de estructuras y colección de metodos de los mismos

Estos son representados graficamente de la siguiente manera.



El caso de uso se explica:

NOMBRE	Suscripcion de sevicio (ingresa Data Entry suscriptor)
DESCRIPCION	El usuario reigstra una nueva suscripción.
ACTORES	Usuario con permisos de operador ESB
PRECONDICIONES	El usuario debe estar autenticado a la aplicación.
FLUJO NORMAL	1. El sistema despliega la interfaz de registro de suscripcion servicio. 2. El usuario ingresa la informacion correspondiente a la nueva suscripcion. 3. El sistema valida la iformacion ingresada y confirma la suscripcion.
FLUJO ALTERNATIVO (DATOS DE ENTRADA ERRONEOS)	1. El sistema confirma una validacion erronea en alguno de los paramDetros. 2. El usuario debe revisar los parametros

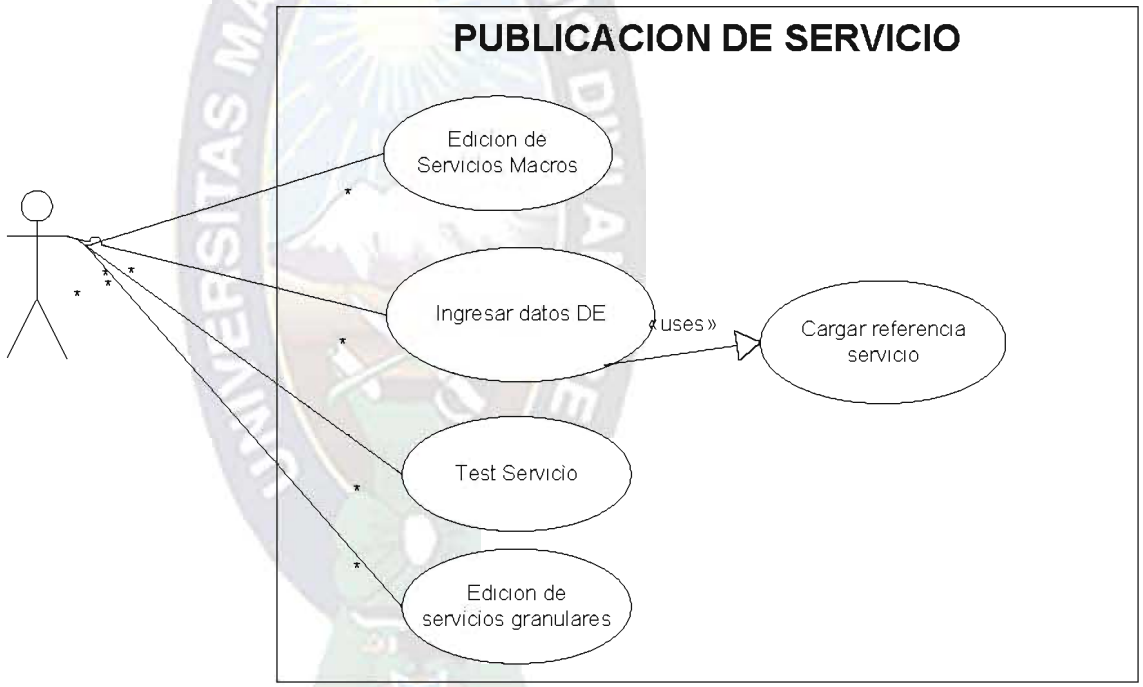
	ingresados
POSTCONDICIONES	Se tendrá un nuevo canal abierto para el consumo del servicio granularizado.

Publicación de un servicio.

El modulo de publicacion de un servicio, presentará las siguientes funcionalidades:

- Creacion de un servicio macro y varios granulares.
- Validacion de la disponibilidad del servicio registrado.
- Test de conexión contra el ESB, elaborando una suscripcion de prueba.

Dichas funcionalidades se reflejan graficamente:



El caso de uso se explica:

NOMBRE	Publicacion de sevicio (ingresa Data Entry suscriptor)
DESCRIPCION	Mediante este caso de uso se puede publicar un servicio de un proveedor previamente registrado.
ACTORES	Usuario con permisos de operador ESB
PRECONDICIONES	El usuario debe estar autentificado con el data entry del modulo de publicacion de servicio.
FLUJO NORMAL	4. El sistema visualiza la interfaz de publicacion de servicio. 5. El usuario ingresa los datos correspondiente al

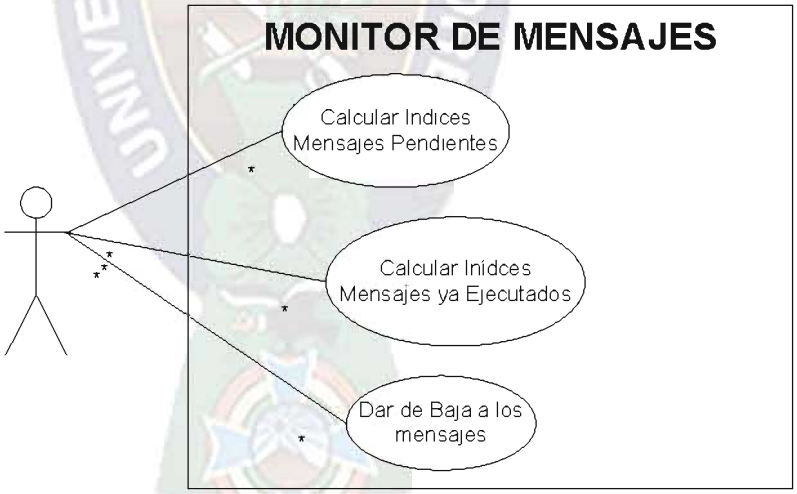
	proveedor y servicio. 6. El sistema valida la informacion ingresada y confirma el registro del servicio.
FLUJO ALTERNATIVO (DATOS DE ENTRADA ERRONEOS)	3. El sistema confirma una validacion erronea en alguno de los parametros. 4. El usuario debe revisar los parametros ingresados
POSTCONDICIONES	Se tendrá registrado un servicio macro y los servicios granularizados..

Monitoreo de mensajes del ESB.

El componente para monitoreo de mensajes tiene por objetivos:

- Realizar el calculo de indices de los mensajes en cola
- Realizar el cálculo de indices de mensajes ejecutados.
- Eliminar mensajes en cola.

Estos son representados graficamente de la siguiente manera.



El caso de uso se explica:

NOMBRE	Monitor de mensajes
DESCRIPCION	El usuario verifica los indices generados por las colas de mensajes.
ACTORES	Usuario con permisos de operador ESB
PRECONDICIONES	El usuario debe estar autenticado a la aplicación.
FLUJO NORMAL	1. El sistema de forma grafica las colas de Ejecucion y la cola de ejecutados.

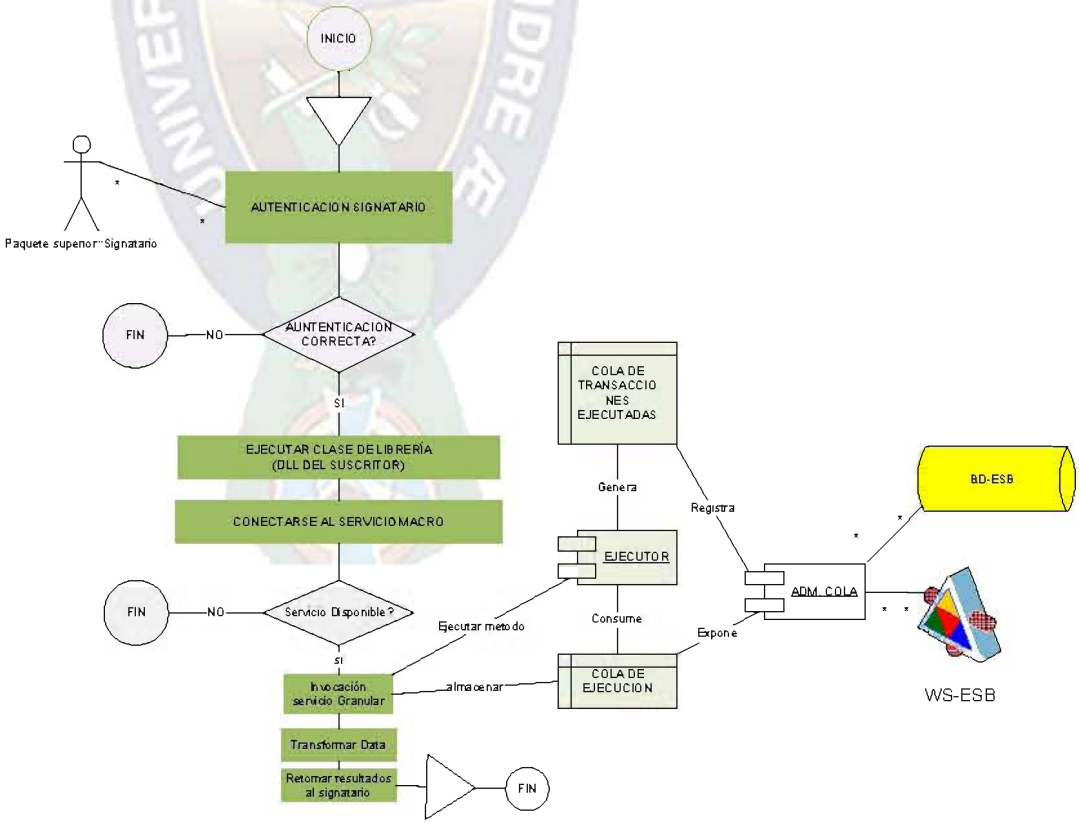
	2. El usuario puede visualizar los indices. 3. El usuario puede detener la ejecucion de la cola, para eliminar mensajes. 4. El sistema presenta los elementos de la cola. 5. El usuario elimina los elementos que desee. 6. Finlaliza la operación y el ESB vuelve a procesar la nueva cola.
POSTCONDICIONES	El usuario dispondra de los indices de atencion.

Consumo de servicio

Los objetivos del modulo son:

- Auntentificar al signatario
- Ver la disponibilidad del canal
- Conectarse con el servicio macro
- Enviar los parametros al servicio granular
- Elborar la transaformacion de los resultados.
- Devolver al signatario el resultado.

El flujo a seguir es este:



El mismo se explica de la siguiente manera:

Descripcion del flujo:

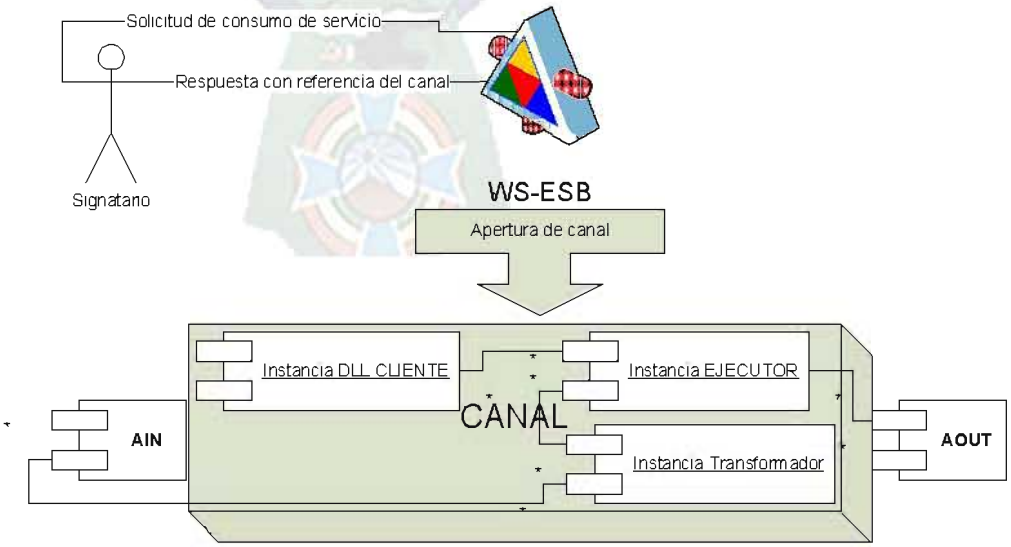
a. El signatario se autentica para el consumo de un servicio granular.
b. Una vez ya verificado la aunteticacion el sistema abre un canal y devuelve una referencia, para que el sisterma se conecta con dicho canal mediante el adaptador de entrada.
c. El canal abierto se contecta con el servicio macro mediante un adaptador de salida.
d. Si existe disponibilidad del servicio entonces se delega al comoponente ejecutor la invocacion del servicio granular y llena la *cola de transacciones ejecutadas*.
e. El administrador de colas realiza caragado de la *cola de ejecucion* y registra los elementos de *cola de transacciones ejecutadas.*
f. Un webservice expone ambas colas.

2. Descripcion de los componentes, en el anterior diagrama se visualiza diferentes componentes, los mismo se explican:

   a. Adm_cola, librería que contiene clases para manipular las colas en base al criterio de prioridad del canal que envia su transaccion.
   b. Ejecutor, librería que invoca de forma dinamica los metodos tanto de la libreia propietaria del signatario, como los motodos web de los servicio.
   c. Bd_ESB, repositorio propio del ESB.
   d. WS-ESB, servicio web que expone las transacciones de las colas.

Interfaces

La interfaz con la que mediara el sginatario es un canal que se abrira cuando se desee consumir un servicio, el mismo se respresenta graficamente asi:

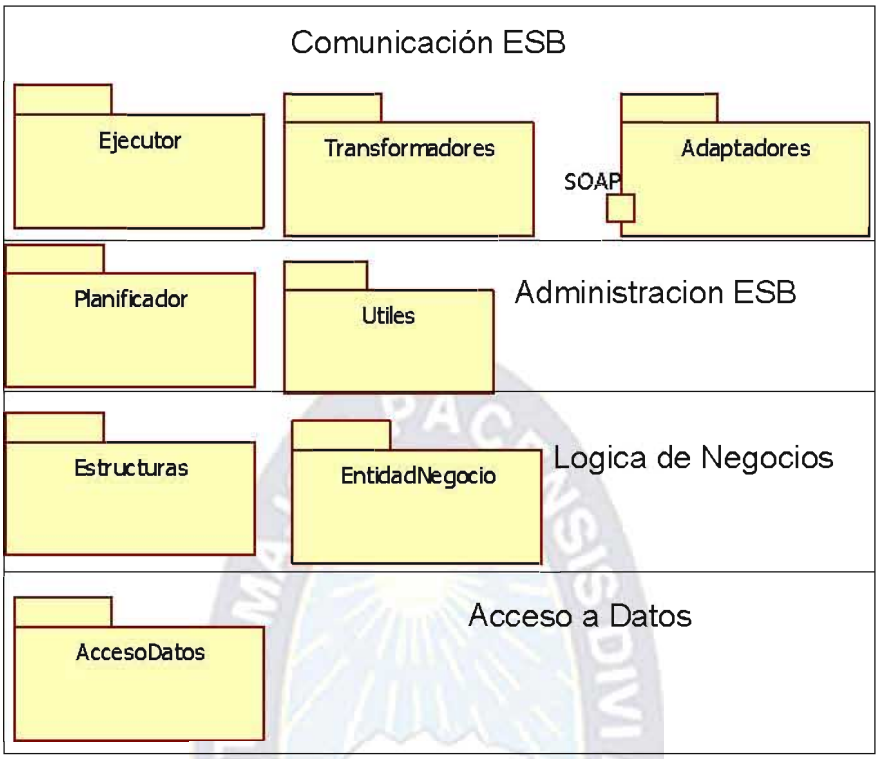


El grafico se explica:

- WS-ESB: servicio al cual se solicitara el consumo de un servicio por parte del signatario.
- Canal: este componente se descubrira al signatario mediante el servicio ws-esb, el mismo cuando tiene una apertura, generara una referencia que sera enviada al signatario para que el se conecta a la nueva instancia del mismo. Para los mismo se debe comprender los componentes internos:
    - AIN: es el adaptador de entrada al canal, este contiene un endpoint que sera al cual el signatario enviara los mensajes para la ejecucion de las transacciones.
    - Instancia DLL CLIENTE (nucleo logico de integracion): componente que instanciara un objecto de la librería del signatario para enviarle las entradas del mensaje, este componente enviara las salidas a la instancia ejecutor.
    - Instancia EJECUTOR: este componente invocara de forma dinamica a los webmotodos de los servicios macros (un servicio granular) el componente AOUT y el resultado de la invocacion sera enviado al componente instancia Transformador.
    - Instancia Transformador: convertira el resultado obtenido en un dataset y lo enviara al cliente nuevamente.

DESCRIPCIÓN Y DIAGRAMA DE COMPONENTES

El proceso de técnico se presenta atraves del siguiente diagrama de paquetes:



Para poder construir el gestor de servicios es necesario elaborar los componentes:

- Acceso a Datos: que tiene como proposito gestionar las conexiones a la Base de datos.
- Logica de Negocios: que tiene por objetivo ser el contenedro de las estructuras para el manejo de errores y otras enumeraciones, además de los objetos del negocio.
- Administracion de ESB: que tiene por objetivo contener a las unidades responsables de la gestion de la publicacion y suscripcion de al ESB.
- Comunicación ESB: que tiene por objetivo contener los componente de comunicación, transformacion e invocacion dinámica del ESB.

Descripción de las unidades de programación

Descripción general de cada unidad

Las unidades de programacion a elaborar son las siguientes:

Nombre del Elemento o Programa	Clase: SIGNATARIO; Método: AddSignatario

Descripción del cambio	
Entradas	1. Nombre (atributo del objeto). 2. Archivo Dll (atributo del objeto). 3. Cant máxima canales (atributo del objeto), en caso de ser 0 se entenderá que no tendrá una cantidad máxima de canales a consumir. 4. Login (atributo del objeto). 5. Password (atributo del objeto). 6. Nombre_clase (atributo del objcto).
Lógica de programación (pseudocódigo)	1. Validar la que la Dll, enviada verificando que la misma contenga una clase con el nombre que contenga el campo Nombre_clase. 2. Si la validación es correcta insertar el objeto signatario a la tabla de la siguiente manera: - Id_signatario: autonumerico. - Nombre: atrubuto nombre del objeto. - Path librería: dirección completa en la cual se cargara el archivo dll que es un atributo del objeto. - Fecha_modificacion: fecha del sistema. - Usr_modificacion: matricula del usuario quien está registrando el objeto - Fecha_alta: fecha del sistema. - Usr_alta: matricula del usuario quien está registrando el objeto. - Login: login del usuario con el cual el signatario consumirá sus cana les, atributo del objeto. - Password: password con el cual el signatario consumirá sus canales, el mismo debe ser encriptado con el algoritmo MD5 - Nombre_clase: nombre de la clase, atributo del objeto signatario. - Habilitado: por defecto debe ser 1. 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	1. Id del signatario registrado.

	2. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD. E0003: Error en Dll a cargar.

Nombre del Elemento o Programa	Clase: SIGNATARIO; Método: EditSignatario
Descripción del cambio	
Entradas	1. IdSignatario (atributo del objeto) 2. Nombre (atributo del objeto). 3. Archivo Dll (atributo del objeto). 4. Cant máxima canales (atributo del objeto), en caso de ser 0 se entenderá que no tendrá una cantidad máxima de canales a consumir. 5. Login (atributo del objeto). 6. Password (atributo del objeto). 7. Nombre_clase (atributo del objeto). 8. Habilitado (atributo del objeto)
Lógica de programación (pseudocódigo)	1. Validar la que la Dll, enviada verificando que la misma contenga una clase con el nombre que contenga el campo Nombre_clase. 2. Si la validación es correcta modificar el objeto signatario a la tabla de la siguiente manera: - Id_signatario: autonumerico. - Nombre: atrubuto nombre del objeto. - Path librería: dirección completa en la cual se cargara el archivo dll que es un atributo del objeto. - Fecha_modificacion: fecha del sistema.

	- Usr_modificacion: matricula del usuario quien está registrando el objeto - Login: login del usuario con el cual el signatario consumirá sus cana les, atributo del objeto. - Password: password con el cual el signatario consumirá sus canales, el mismo debe ser encriptado con el algoritmo MD5 - Nombre_clase: nombre de la clase, atributo del objeto signatario. - Habilitado: atributo del objeto. 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	3. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0003: Error en Dll a cargar. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: SIGNATARIO; Método: BajaSignatario
Descripción del cambio	
Entradas	1. IdSignatario (atributo del objeto)
Lógica de programación	1. Desplegar solo los signatarios que estén habilitados. 2. Si la validación es correcta modificar el

(pseudocódigo)	objeto signatario a la tabla de la siguiente manera: - habilitado: se le asigna el valor 0 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	4. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: AltaProveedor
Descripción del cambio	
Entradas	1. Nombre (atributo objeto) 2. Descripción (atributo objeto) 3. Usr_alta: matricula del usuario q registro al objecto proveedor, 4. Fecha_alta: fecha del sistema. 5. Usr_registro: matricula del usuario q registro al objecto proveedor, 6. Fecha_registro: fecha del sistema.
Lógica de programación (pseudocódigo)	4. Verificar los dato ingresados. 5. Si la validación es correcta ingresar el objeto provedor a la tabla de la siguiente manera: - Id_proveedor: autonumerico.

	- Nombre: atributo del objeto. - Descripción: atributo del objeto. - Usr_alta: atributo del objeto. - Fecha_alta: atributo del objeto. - Usr_modificacion: atributo del objeto. - Fecha_modificacion: atributo del objeto. 6. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	5. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: EditProveedor
Descripción del cambio	
Entradas	1. Id_proveedor. 2. Nombre (atributo objeto) 3. Descripción (atributo objeto) 4. Usr_alta: matricula del usuario q registro al objecto proveedor, 5. Fecha_alta: fecha del sistema. 6. Usr_registro: matricula del usuario q registro al objecto proveedor,

	7. Fecha_registro: fecha del sistema.
Lógica de programación (pseudocódigo)	1. Verificar los datos ingresados. 2. Si la validación es correcta modificar el objeto proveedor a la tabla de la siguiente manera: - Id_proveedor atributo del objeto. - Nombre: atributo del objeto. - Descripción: atributo del objeto. - Usr_alta: atributo del objeto. - Fecha_alta: atributo del objeto. - Usr_modificacion: atributo del objeto. - Fecha_modificacion: atributo del objeto. 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	6. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: AltaServicioMacro
Descripción del cambio	
Entradas	1. Descripción(atributo objeto) 2. referencia(atributo objeto) 3. Usr_alta: matricula del usuario q registro al objecto

	proveedor, 4. Fecha_alta: fecha del sistema. 5. Usr_registro: matricula del usuario q registro al objecto proveedor, 6. Fecha_registro: fecha del sistema. 7. Id_proveedor: atributo del objeto.
Lógica de programación (pseudocódigo)	1. Verificar los datos ingresados. 2. Si la validación es correcta modificar el objeto proveedor a la tabla de la siguiente manera: - Id_proveedor autonumérico. - Descripcion: atributo del objeto. - referencia: atributo del objeto. - Usr_alta: atributo del objeto. - Fecha_alta: atributo del objeto. - Usr_modificacion: atributo del objeto. - Fecha_modificacion: atributo del objeto. - Habilitado: 1 - Id_proveedor: atributo del objeto 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	7. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: BajaServicioMacro
Descripción del cambio	
Entradas	1. Id_proveedor (atributo objeto)
Lógica de programación (pseudocódigo)	1. Verificar los datos ingresados. 2. Si la validación es correcta modificar el objeto proveedor a la tabla de la siguiente manera: - habilitado 0. 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	8. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: AltaServicioAtomico
Descripción del cambio	
Entradas	1. Nombre_firma (atributo objeto) 2. referencia(atributo objeto) 3. Usr_alta: matricula del usuario q registro al objecto proveedor, 4. Fecha_alta: fecha del sistema. 5. Usr_registro: matricula del usuario q registro al objecto proveedor,

	6. Fecha_registro: fecha del sistema. 7. Id_serv_macro: atributo del objeto.
Lógica de programación (pseudocódigo)	1. Verificar los datos ingresados. 2. Si la validación es correcta modificar el objeto proveedor a la tabla de la siguiente manera: - Id_serv_atomico: autonumérico. - Descripcion: atributo del objeto. - Nombre_firma: atributo del objeto. - Usr_alta: atributo del objeto. - Fecha_alta: atributo del objeto. - Usr_modificacion: atributo del objeto. - Fecha_modificacion: atributo del objeto. - Habilitado: 1 - Id_serv_macro: atributo del objeto 3. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	9. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	
Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Nombre del Elemento o Programa	Clase: PROVEEDOR; Método: AltaServicioAtomico
Descripción del	

cambio	
Entradas	8. Nombre_firma (atributo objeto) 9. referencia(atributo objeto) 10. Usr_alta: matricula del usuario q registro al objecto proveedor, 11. Fecha_alta: fecha del sistema. 12. Usr_registro: matricula del usuario q registro al objeto proveedor, 13. Fecha_registro: fecha del sistema. 14. Id_serv_macro: atributo del objeto.
Lógica de programación (pseudocódigo)	4. Verificar los datos ingresados. 5. Si la validación es correcta modificar el objeto proveedor a la tabla de la siguiente manera: - Id_serv_atomico: autonumérico. - Descripcion: atributo del objeto. - Nombre_firma: atributo del objeto. - Usr_alta: atributo del objeto. - Fecha_alta: atributo del objeto. - Usr_modificacion: atributo del objeto. - Fecha_modificacion: atributo del objeto. - Habilitado: 1 - Id_serv_macro: atributo del objeto 6. Una vez realizada la transacción enviar un mensaje de éxito a la interfaz.
Salidas	10. Mensaje de error o éxito de la transacción.
Interfaces usadas	
Otras consideraciones	
Requisito (s) Funcional(es) que se implementa.	Administracion genérica
Otros procesos Ad-Hoc impactados	

Alertas y Mensajes de Retorno	Los errores enviados serán E0001: Error en parámetros enviados para el registro. E0002: Error por conexión en BD.

Diagrama de Estructural.

Capa de Lógica de negocios: la misma se compone de los siguiente elementos

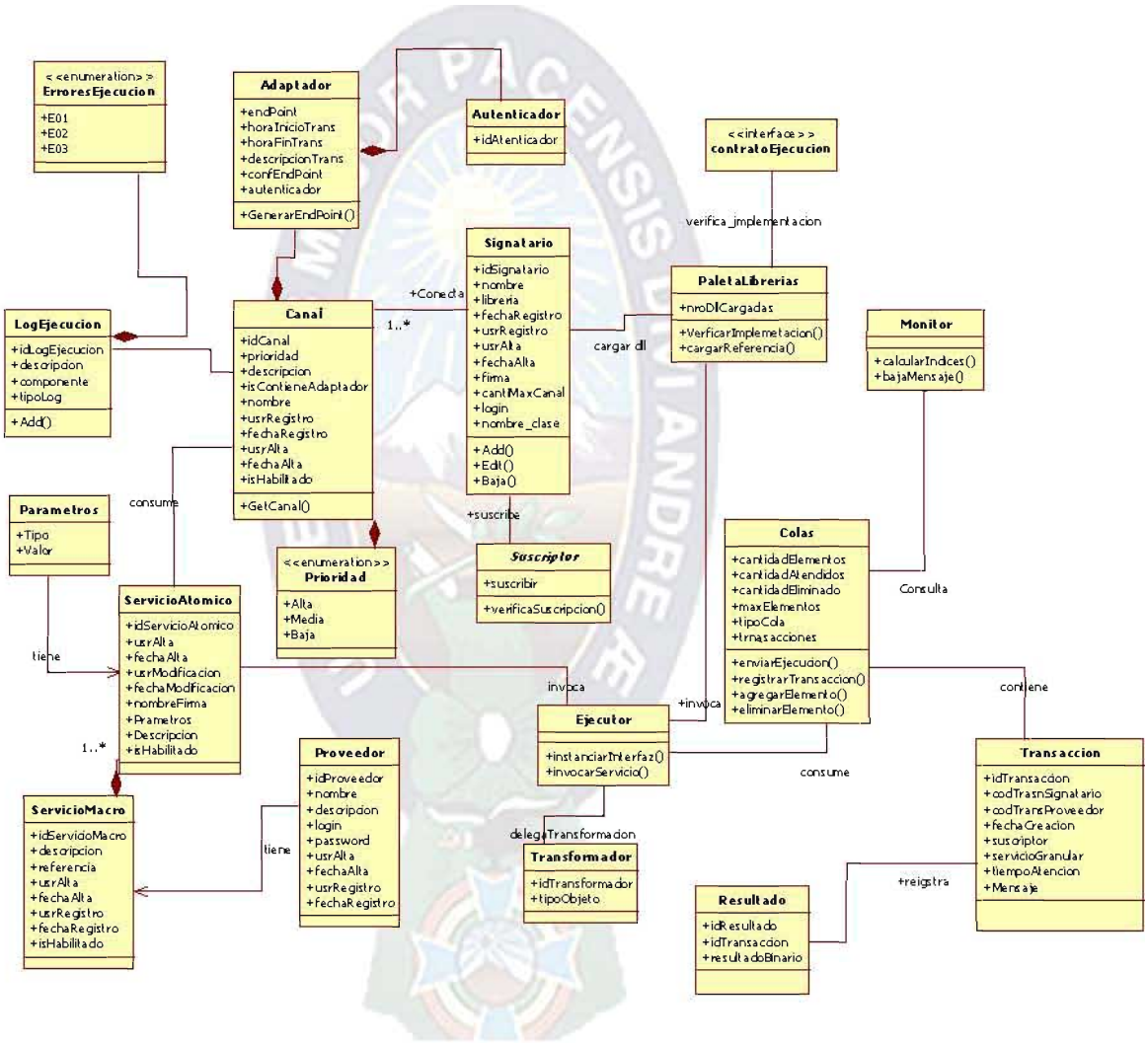
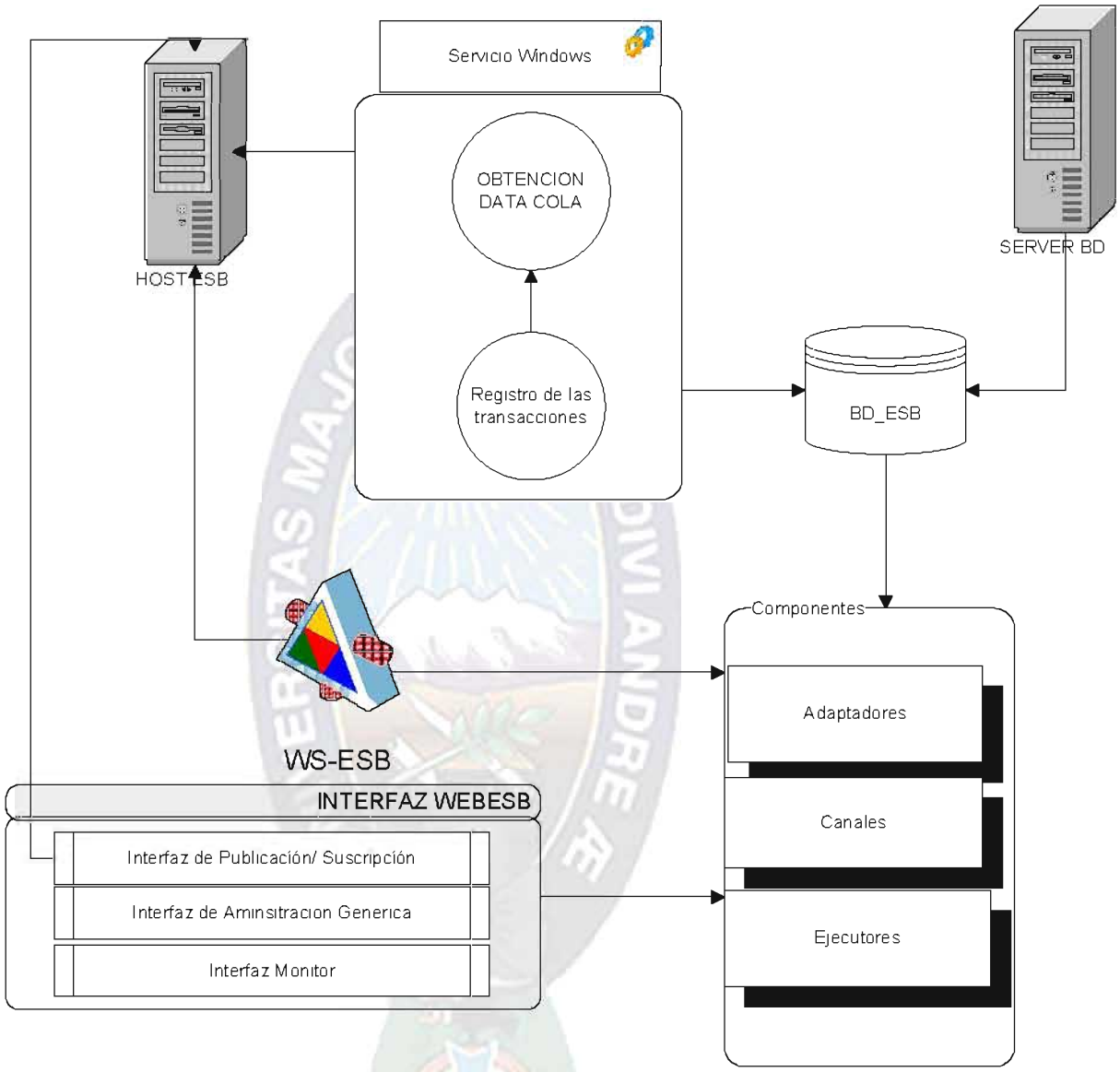


Diagrama de Relación/Implementación Componentes Lógicos y Físicos.



La división de los componentes físicamente será:

- HOST Físico para los siguientes componentes:
    - Interfaz Web.
    - Servicio Windows.
    - Web Services.

- HOST Físico para:
    - Base de datos.

CAPITULO III

ANALISIS DE RESULTADOS

III. ANALISIS DE RESULTADOS

1. CONCLUSIONES GENERALES

Actualmente el uso de un ESB, es confundido con otros conceptos como EAI o ESA; por la naturaleza de su arquitectura, podemos definir como conclusiones aclarativas las siguientes:

- Un ESB no corresponde a un ruteo de red inteligente, ya que posee propiedades dentro del ámbito de Arquitectura de la Información de lógica comercial que le permiten definir aspectos de los procesos del flujo comercial.

- Los patrones de diseño que fueron implementados para la construcción del ESB son correspondientes a un ámbito netamente técnico y no así a solo a obedecer modelos teóricos; ya que se determina como una alternativa a solucionar problemas técnicos de los servicios IT en una organización.

- La escalabilidad de un ESB está limitada por la infraestructura de la institución ya que su performance tiene un comportamiento en escala vertical, es decir que para mejor rendimiento no solo requiere de memoria si no de mucho procesador.

- El ESB puede ser implementado en cualquier ambiente comercial donde exista un ecosistema de servicios empresariales que requiere comunicarse. Teniendo posibilidades de ingresar incluso a ambientes de sistemas industriales donde la cadena de producción es muy distinta a un flujo comercial empresarial.

- ESB no es un patrón de diseño sino más bien una herramienta de gestión de servicios.

- Un ESB tiene más impacto en una institución cuando la red de comunicación de servicios es amplia

2. ESTADO DE LOS OBJETIVOS

- ***Flexibilidad a cambios en el flujo comercial:*** a través de la gestión de los servicios que son la automatización procesos de negocio podemos tener una clara visión de cómo interactúa el flujo comercial en la área de intercambio, minimizando el impacto de un cambio en el mismo en los aplicativos empresariales de una organización.

- ***Integración de sistemas construidos en diferentes plataformas:*** a través del componente adaptador del Bus de servicio empresarial logramos cumplir con este objetivo.

- ***Reutilización de aplicativos LOB:*** mediante el patrón de suscripción podemos adquirir canales reutilizables dentro del ESB.

- ***Transparencia en el consumo y alojamiento de los Servicios, liberando al consumidor la ubicación física (hosting) del proveedor del servicio:*** A través del modelo del dominio que se presenta para el ESB es posible consumir el ESB desde cualquier área física que tenga conexión con el servidor de hosteo, sin embargo es recomendable que eta interacción sea dentro de la DMZ.

- ***Conversión de Protocolo y Transporte, liberando a los consumidores de la tarea de socialización y marsahlling al momento de consumir un servicio:*** A través de los componentes canal y transformador, podemos determinar la forma de serializacion, que tendrá el canal (estas tareas se definen en el behavior como aclarion técnica).

- ***Ruteo de Mensajes, definiendo a donde enviar el mensaje a través de parámetros obtenidos del proveedor.*** Básicamente esta es una de las tareas fundamentales del componente canal de una ESB.

- ***Seguridad y Monitoreo, permitiendo centralizar las funciones de inscripción, autenticación y autorización de los Mensajes; además de proporcionar. Además de proporcionar un ambiente de administración y control de los flujo de mensajes****. Este aspecto es cubierto con el Data Entry propuesto para la administración de la cola.*

- ***Estandarización dentro del intercambio de mensajería.*** En este punto los papeles de los núcleos lógicos que determinaran los adaptares a usar cuando el signatario consuma el canal del ESB, permitiendo un alto grado de covarianza en el comportamiento del ESB.

3. PRUEBA DE HIPOTESIS

Por la naturaleza de la tesis descriptiva no se realizara una prueba de hipótesis estadística, ya que las variables y factores implícitos de la hipótesis son cualitativos:

"Mediante la implementación de un entorno de integración de servicios, se puede obtener integridad transaccional, flexibilidad, escalabilidad y mejoras en el flujo comercial de una organización, además de una gestión controlada de políticas sobre un ecosistema de servicios caóticos en una Institución"

Sin embargo para verificar la valides de la investigación usaremos los indicadores de las variables identificadas en la hipótesis y los cuantificaremos:

VARIABLE	INDICADOR	RESULTADOS
Entono de integración de servicios	Nivel de integración que ofrece a un conjunto de sistemas	Está en función a los aplicativos que se suscriban al ESB, si el total de ellos son parte del mismo es posible integrar los aplicativos al 100%
Integridad Transaccional	Calidad de las transacciones y de información proporcionadas por los servicios	La arquitectura proporcionada por el ESB permite la alta disponibilidad y la agregación de información mediante los núcleos lógicos de los canales.
Flexibilidad de mejoras sobre los servicios	Nivel de desacoplamiento de los servicios	Conceptualmente a través de EAI, generamos una área de intercambio, la misma que tiene por objetivo llegar a un 0% de acoplamiento en los sistemas.
Gestión de reglas políticas	Porcentaje de cumplimiento de las normas impuestas en la institución.	A través de la gestión de los servicios podemos fijar políticas de accesos, ejecución y seguimiento de los procesos de negocio automatizados
Flujo Comercial	Complejidad de la totalidad de los procesos de negocio	La complejidad del proceso de negocio puede o no reducir, sin embargo a través de un ESB es posible controlar el impacto de los posibles cambios que sufran los mismos.

4. BENEFICIOS DE LA INVESTIGACION

Antes de evaluar los beneficios que infieren el uso de un ESB, veamos algunas aspectos negativos que sufren actualmente las organización cuando automatiza sus procesos:

- Los procesos de negocio tiene dificultad en evolucionar por las adecuaciones que una red de servicios contraponen.
- El ESB soluciona problemas de integración a nivel granular, sin embargo la gestión cada proceso lógico que se aloja en si mimos requiere de un constante mantenimiento.
- La construcción de distintos adaptadores para diferentes protocolos reducen costos en el desarrollo de interfaces entre los aplicativos, causa de esto es la heterogeneidad de plataformas tecnologicas.

- Para la implementación de un ESB es necesario tomar en cuenta:

    - La cantidad de servicios que se desea integrar, ya que esta pueda inferir en costos de infraestructura.
    - El nivel de acoplamiento entre los mismos, posiblemente sea mejor rehacer algunos aplicativos para que estos puedan entrar en el esquema del ESB.
    - La diferencia tecnológica entre los servicios, esto puede requerir construcción de adaptadores por ejemplo para el protocolo MQ.

Una vez mencionados estos aspectos, pasemos a listar los beneficios:

- Baja de acoplamiento entre procesos comerciales de una organización, beneficiando al flujo comercial, mediante la independencia del mecanismo de intercomunicabilidad entre los mismos.

- Independencia de comunicabilidad de los aplicativos empresariales, evitando en gran medida el acoplamiento de los mismos mejorando la calidad de los procesos del flujo comercial. El diseño del ESB permite la independencia de los protocolos de comunicación además del manejo de concurrencia.

- Encapsulación del área de intercambio de información que presenta la lógica comercial, se entiende que toda la información que será administrada por el ESB se presentara como una nueva capa Empresarial dentro de la arquitectura de todo el flujo comercial.

- Extensibilidad del modelo orientado a SOA, mejorando el diseño de los procesos comerciales. A través de Oslo nos permitimos extender SOA a nivel de modelo, mediante la posibilidad del rediseño de procesos independientemente del impacto técnico que representaría estos cambios en otros aplicativos.

- Alta productividad en el diseño de las interfaces de los procesos de negocio, a través de los contratos de los que presenta cada unidad lógica del signatario del ESB.

- Posibilidad de intercomunicación de aplicaciones en un ambiente de alta disponibilidad, por la forma de alojamiento que requiere el ESB, este puede estar presente en distintos puntos de la infraestructura de una institución, además de poder distribuir las transacciones en distintos repositorios simultáneamente.

Finalmente por todo lo mencionado podemos afirmar que:

"Un ESB es proporciona un ambiente de integración a ecosistemas caóticos, que en su comportamiento evitan que el flujo comercial evolucione"

5. RECOMENDACIONES

La concepción de un ESB nace con la necesidad inicial de permitir evolucionar a los procesos tecnológicos para cubrir todas las necesidades de los procesos de un flujo comercial.

Actualmente el desarrollo de aplicaciones tiende a ser artesanal, sin seguir un modelo de análisis que beneficie de forma productiva a los componentes del proyecto, incurriendo en varios factores que mitigan tanto la explotación como la construcción de dichos aplicativos; por lo cual se recomienda que en el momento de construir el dominio del modelo se tomen en cuenta siempre el diseño del proceso de negocio en sí, con la ayuda de expertos en la cadena productiva a la cual corresponda el sistema a desarrollar y de esta forma ser más flexible a los posibles cambios, con el objeto de ser parte de un ecosistema integrado y ordenado. Ya que considerar ser signatario de un ESB se que por lo general siempre trae como consecuencia posibles actividades de reingeniería; pues este recurso está preparado para servicios que son semánticamente representativos para la lógica empresarial.

Finalmente la tarea del monitoreo de calidad de integración de los aplicativos es una tarea constante de un profesional de IT.

IV. BIBLIOGRAFIA

Titulo: The Art of Software Architecture: Design Methods and Techniques

Autor: Stephen T. Albin

Detalles: ISBN:0471228869, John Wiley & Sons © 2003 (312 pages)

Titulo: Patterns of Enterprise Application Architecture

Autor: Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford

Detalles: Publisher: Addison Wesley
Pub Date: November 05, 2002
ISBN: 0-321-12742-0
Pages: 560

Titulo: Arquitectura de aplicaciones .NET: Diseño de aplicaciones y servicios

Autor: Patterns & Practices

Detalles: Microsoft Corporation

Diciembre de 2002

Titulo: Arquitectura de aplicaciones .NET: Diseño de aplicaciones y servicios

Autor: Patterns & Practices

Detalles: Microsoft Corporation

Diciembre de 2002

Titulo: Integración de Aplicaciones de Empresa(EnterpriseApplicationIntegratiónEAI)

Autor: Pedro Álvarez

Detalles: Universidad de Zaragoza 2002

Titulo: The Architecture Journal - Model Driven SOA with OSLO

Autor: Cesar de la Torre

Detalles: Microsoft 2010

Titulo: The Architecture Journal - The Internet Service Bus

Autor: Donald F. Ferguson, Dennis Pilarinos, John Shewchuk

Titulo: The Architecture Journal - Model Driven SOA with OSLO

Autor: Cesar de la Torre

Detalles: Microsoft 2010

Titulo: Fundamentos e Implementación de Arquitectura Orientada a Servicio SOA

Autor: Alberto Meléndez

Detalles: GBM Corporation

Titulo: Enterprise SOA: Designing IT for Business Innovation

Autor: By Thomas Mattern, Dan Woods

Detalles:
Publisher: **O'Reilly**
Pub Date: **April 2006**
Print ISBN-10: **0-596-10238-0**
Print ISBN-13: **978-0-59-610238-8**
Pages: **452**

DOCUMENTOS